

DTIC FILE COPY

1

AD-A220 147



DTIC
ELECTE
APR 05 1990

S C D

Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

DISTRIBUTION STATEMENT A

Approved to: public release
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

90 04 05 131

AFIT/GCE/ENG/90M-2

A COMPARISON OF A RELATIONAL
AND NESTED-RELATIONAL IDEF₀
DATA MODEL

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Gerald Roger Morris, B.S.E.E.

Captain, USAF

MARCH, 1990

Approved for public release; distribution unlimited

AFIT/GCE/ENG/90M-2

1

DTIC
REF ID: A62510
S 0
D

A COMPARISON OF A RELATIONAL
AND NESTED-RELATIONAL IDEF₀
DATA MODEL

THESIS

Gerald Roger Morris
Captain, USAF

AFIT/GCE/ENG/90M-2

Approved for public release; distribution unlimited

Preface

The purpose of this thesis is to develop an abstract data model of a computer aided software engineering (CASE) methodology, and to compare the query complexity, database size, and speed of query execution of a relational database management system (DBMS) implementation of the methodology with a nested-relational DBMS implementation. The United States Air Force Integrated Computer Aided Manufacturing (ICAM) program defines a subset of Ross's Structured Analysis (SA) language called ICAM Definition Method Zero (IDEF₀); it is precisely this IDEF₀ subset that is considered (30)(25). Ingres Corporation's relational DBMS, Ingres, is the implementation media for the relational version; the University of Wisconsin's Extensible Object-oriented Database, Exodus, is the implementation media for the nested-relational version (29)(4).

The comparison is undertaken to demonstrate potential advantages of a nested-relational DBMS for this application. Additionally, the development of an abstract data model for IDEF₀ is directly related to on-going AFIT research efforts associated with the Strategic Defense Initiative Organization (SDIO), which has adopted the IDEF₀ analysis language.

I extend my gratitude to several people who supported me during this effort. I thank my thesis advisor, Major Mark Roth for his patience and guidance in the area of database theory and application. I thank the other two members of my committee, Dr. Thomas Hartrum, who strongly influenced the development of the abstract data model, and Dr. Gary Lamont, who taught me early on to assume nothing and to return to first principles when lost. I thank Capt Neal Smith and Capt Ken Austin, who helped me develop the abstract data model. I thank Capt Mike Mankus who developed the nested-relational DBMS and Capt Jim Kirkpatrick for his insight in helping me develop the nested-relational version. I especially thank Penny for her moral support during these past months.

Gerald Roger Morris

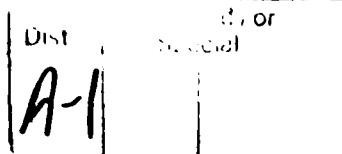
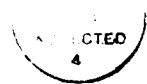


Table of Contents

	Page
Preface	ii
Table of Contents	iii
List of Figures	viii
List of Tables	x
Abstract	xi
I. INTRODUCTION	1
General Issues	1
Background	2
Overview of CASE Development	2
Overview of Database Development	4
Problem Statement	10
Plan of Attack	11
Development of an Abstract Data Model	11
Mapping of Abstract Data Model	13
Example Database Instance	13
Development of Queries	13
Comparison	14
Scope and Limitations	15
Scope	15
Limitations	15
Sequence of Presentation	16

	Page
II. LITERATURE REVIEW	17
Introduction	17
Overview of IDEF₀	17
AFIT and IDEF₀	21
Overview of Exodus	22
CASE Tools and DBMS	23
An Analysis Scenario - The Need for a DBMS.	24
Mapping Difficulties.	27
Commercial Databases for CASE Tools.	30
Integration of CASE Tools.	31
Summary	35
III. METHODOLOGY	36
Introduction	36
Extended E-R Notation	36
IDEF₀ Abstract Data Model	37
Essential Data Model.	39
Drawing Data Model.	44
IDEF₀ Relational Database	49
Design Trade Offs.	49
Relational Design.	50
Example Relational Database Instance.	55
Relational Implementation.	55
SQL Queries.	58
IDEF₀ Nested Relational Database	61
Design Trade-Offs.	61
Nested-Relational Design.	63
Example Nested-Relational Database Instance.	71

	Page
Nested-Relational Implementation.	72
SQL/NF Queries.	72
Summary	74
IV. FINDINGS	75
Introduction	75
Query Complexity	75
A Definition of Complexity.	75
Comparison of SQL versus SQL/NF.	77
Size of Database	78
Relational Logical Size.	78
Nested-Relational Logical Size.	79
Speed of Query Execution	83
Disk Resident Project Data.	83
Memory Resident Project Data.	84
Summary	91
V. CONCLUSIONS AND RECOMMENDATIONS	92
Introduction	92
Summary	92
Conclusions	93
Recommendations	95
Appendix A. Some CASE Tools and Vendors	97
Appendix B. IDEF ₀ Language Features	100
Appendix C. SAtool Products	101
Typical SAtool IDEF ₀ Drawing Outputs	101
Data Dictionary Outputs	104

	Page
ACTIVITY Data Dictionary.	104
DATA ELEMENT Data Dictionary.	105
Appendix D. Analysis Phase Data Base	106
Appendix E. Typical Data Manager Session	109
Appendix F. Example IDEF ₀ Relational Database Instance	111
Appendix G. SQL Scripts	117
Create Tables	117
Load Database	120
Erase Database	121
Show Database	122
Extract Drawing Data	123
A-0 Drawing Data.	123
A0 Drawing Data.	127
Extract Essential Data	137
Activity Data Dictionary.	137
Data Element Data Dictionary.	139
Appendix H. Example IDEF ₀ Nested-Relational Database Instance	142
Appendix I. SQL/NF Scripts	158
Create Tables	158
Load Database	160
Erase Database	161
Show Database	161
Extract Drawing Data	161
Extract Essential Data	164
Activity Data Dictionary.	165

	Page
Data Element Data Dictionary	166
Appendix J. Ada Package for Drawing Data Structures	167
Bibliography	170
Vita	172

List of Figures

Figure	Page
1. Typical Hierarchical Database	5
2. Typical Network Database	6
3. Typical Relational Database	7
4. Typical Entity-Relationship Diagram	9
5. IDEF ₀ Abstract Data Model	12
6. Structured Decomposition	18
7. A-0 Diagram	19
8. A0 Diagram	20
9. SAtool Products	22
10. Highly Simplified E-R Diagram of SAtool Data Model	29
11. Common Database System Structure	34
12. Modified Entity-Relationship Notation	37
13. IDEF ₀ Abstract Data Model: Essential Data and Drawing Data	38
14. IDEF ₀ ACTIVITY Essential Data Model	40
15. IDEF ₀ DATA ELEMENT Essential Data Model	41
16. IDEF ₀ ACTIVITY Drawing Data Model	45
17. IDEF ₀ DATA ELEMENT Drawing Data Model	46
18. A-0 Diagram (partial drawing 1)	59
19. A-0 Diagram (partial drawing 2)	60
20. SAtool Products	101
21. Typical A-0 Diagram	102
22. Typical A0 Diagram	103
23. A-0 Diagram (partial drawing 3)	124
24. A-0 Diagram (partial drawing 4)	126
25. A0 Diagram (partial drawing 1)	128

Figure	Page
26. A0 Diagram (partial drawing 2)	130
27. A0 Diagram (partial drawing 3)	131
28. A0 Diagram (partial drawing 4)	135

List of Tables

Table	Page
1. Description of Components in the Essential Data Model	42
2. Description of Components in the Drawing Data Model	47
3. Relational Design	51
4. Mapping of E-R Essential Data to Relational Design	56
5. Mapping of E-R Drawing Data to Relational Design	57
6. Nested-Relational Design	64
7. Comparison of Query Script Complexity	77
8. Comparison of DDL/DML Script Complexities	78
9. Simple Relational Example	79
10. Simple Nested-Relational Example	79
11. Logical Size of Relational Instance	81
12. Logical Size of Nested-Relational Instance	82
13. Relative Query Speeds: Number of Disk Accesses	84
14. IDEF ₀ Language Features	100

Abstract

This thesis develops an abstract data model of a particular computer aided software engineering (CASE) methodology, and compares the query complexity, database size, and speed of query execution of a relational database management system (DBMS) implementation of the methodology with a nested-relational DBMS implementation of the same CASE methodology. In particular, the thesis considers the United States Air Force Integrated Computer Aided Manufacturing (ICAM) program's subset of Ross's Structured Analysis (SA) language called ICAM Definition Method Zero (IDEF₀). Ingres Corporation's relational DBMS, Ingres, is the implementation media for the relational version. The University of Wisconsin's extensible database, Exodus, is the implementation media for the nested-relational version.

The thesis provides background information on the development of CASE methodologies and the development of database management systems. Additionally, it provides an overview of the IDEF₀ analysis language, and the Exodus extensible DBMS.

Included in the thesis is an abstract data model of the IDEF₀ language. The model partitions IDEF₀ into an essential data model and a drawing data model. This partitioned representation facilitates ongoing and future research relative to syntax checking, generation of an executable software specification, and automatic layout of SA diagrams. Since IDEF₀ is the analysis methodology selected by the Strategic Defense Initiative Organization, the abstract data model alone is of importance.

The abstract data model is mapped into a relational representation and implemented within Ingres. The relational representation is mapped into a nested-relational representation and implemented within Exodus. The two implementations are compared to see if there are any advantages to be gained by using a nested-relational DBMS for this type of application (CASE tool data). The areas of comparison include query complexity, size of the database, and speed of query execution.

A COMPARISON OF A RELATIONAL AND NESTED-RELATIONAL IDEF₀ DATA MODEL

I. INTRODUCTION

General Issues

The application of the modern digital computer was the stepping-stone from which software engineering and database theory developed. This thesis investigation involves both software engineering and database management systems (DBMS). It compares a relational DBMS implementation of a software engineering methodology with a nested-relational DBMS implementation of the same methodology. The United States Air Force Integrated Computer Aided Manufacturing (ICAM) program defines a subset of Ross's Structured Analysis (SA) language called ICAM Definition Method Zero (IDEF₀); it is this language which is considered in this research (25) (30). The relational version of the IDEF₀ language is implemented within Ingres Corporation's relational DBMS, Ingres (29). The nested-relational version is implemented within the University of Wisconsin's Extensible Object-oriented Database, Exodus (4).

The comparison is undertaken to determine potential advantages relative to query complexity, size of the database, and speed of query execution of a nested-relational DBMS for the application of computer aided software engineering (CASE) tools. Additionally, the development of an abstract data model for the IDEF₀ language is directly related to on-going AFIT research efforts associated with the Strategic Defense Initiative Organization (SDIO), which has adopted the IDEF₀ analysis language.

Background

Overview of CASE Development. Software engineering is a relatively new field which has undergone dramatic transformation in the past 40 years. In the early years, computer programming and software development in general was pretty much a "black art" which depended upon the skill of a few "high priests." There were often cost and schedule overruns. As Betty Forman said, "If carpenters built buildings the way programmers write programs, the first termite would destroy civilization" (13:53). The community recognized this problem and began to address it. A workshop in 1968, in Garmisch, West Germany, and a subsequent one in Rome, Italy in 1969 looked at the growing technical and managerial problems associated with the development and maintenance of computer software. According to Fairley, it was these workshops which coined the phrase, software engineering (12:4). Fairley proposes the following definition for software engineering:

Software engineering is the technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates. (12:4)

There are literally dozens of software engineering approaches covering virtually all phases of software development. Some the theoretical work concentrated on the requirements analysis phase. Larry Constantine invented data flow diagrams and perhaps structured programming; one of the first books on structured design was written by Constantine and Edward Yourdon (41). Tom DeMarco wrote the classic book on the principles of structured analysis and showed the use of data flow diagramming as part of a software analysis methodology (11). Paul Ward and Steve Mellor did important work in the area of real-time structured analysis; Ward's 1985 paper describes the extensions made to DeMarco's data flow diagram, which allow it to represent timing and control information (40). In 1987, Paul Ward, Hughes Aircraft Company's Randall Jensen, Honeywell's William Bruyn, and Boeing's Dinesh Keskar, developed the Extended System Modeling Language

(ESML),¹ a method that combines some features of both the Hatley and Ward-Mellor methods (18).

Roger Pressman gives an overview of some other approaches (27). For example, Jackson System Development (JSD) combines a natural language during the initial phases with structure charts and structured text added during the later phases. Another methodology is the Warnier-Orr Data Structured Systems Development (DSDD), which considers data hierarchy, information flow, and functional characteristics. There are several methodologies based upon some type of requirements language, e.g., Software Requirements Engineering Methodology (SREM), and Problem Statement Language/Problem Statement Analyzer (PSL/PSA). Another increasingly popular approach is the so-called object-oriented methodology, which attempts to link real-world objects and their associated operations.

Of particular interest, is Ross's Structured Analysis (30). This graphical analysis language is the basis for IDEF₀, the language modeled in this investigation (25). Like the traditional data flow diagram, SA is a hierarchical decomposition. It allows data (nouns) and activities (verbs) to be modeled via arrows, boxes, and other graphical and textual devices. Additional information about SA and IDEF₀ is given in Chapter II of this thesis.

The early theoretical work naturally led to the development of computer based tools to assist the software developers. The term computer aided software engineering (CASE) is used to describe these computer based tools. First-generation CASE tools were developed using a theoretical and technological base that changed even as the tools were being written. To a large extent this is still true. Many of today's CASE products are constantly being revised to accommodate new theories, methodologies, and programming languages. CASE has become a grab bag for a variety of software products and services; some are quite valuable, other are just "pieces of methodology out chasing markets" (17:52). A list of some commercially available CASE tools is included in Appendix A.

¹While assigned to the Government Plant Representative Office at Hughes, I had the opportunity to review Dr. Jensen's draft paper on the ESML model.

A software development organization needs a suite of CASE tools rather than one "super-tool." In fact any one tool that tried to combine all features would be rather unwieldy. Hawley expresses this sentiment rather succinctly:

To combine all the functions and features known to CASE tools and designate that list as the standard is a disservice to users and vendors. It makes the ideal CASE product resemble an elephant; enormous, clumsy, frightened, and expensive. (17:53)

Since each of the tools within a CASE environment must be able to use the data generated by the other tools, it is important to develop a formal or semi-formal model for each methodology being automated and create an environment in which each of these models can exchange data, perhaps through some type of DBMS.

Overview of Database Development. Database system theory is even more fledgling than computers and software in general. In the thirty years since 1960, it has experienced dramatic changes. Businesses took advantage of the power and speed of computers; the complexity and size of data processing applications began to grow. These applications were based on a file-processing system wherein the various pieces of information were stored in separate files. Korth and Silberschatz point out that this file-based approach has major disadvantages such as data redundancy, difficulty in accessing data, concurrency problems due to multiple users, and security problems (20:2). Companies such as IBM, North American Aviation (now Rockwell International), and General Electric had extensive database requirements that were rapidly exceeding the capability of the existing programmer community. It was becoming difficult for these large organizations to complete existing projects, let alone take on new projects. Increasing amounts of time were being spent on special purpose code to accommodate multiple users, provide adequate security, ensure the integrity of data, and so forth. Each new application basically reinvented the wheel in terms of data structures, access methods, and so forth. As a consequence of these issues, database management systems (DBMS) were developed. As far as traditional business applications are concerned,

database system theory and practice appears to have matured and stabilized. "The fundamental database system concepts are now well defined and well understood" (20:xiii).

The first DBMS systems were based upon the hierarchical model. In this logical approach, records are contained in multiple levels that graphically form a tree structure with the root at the top and the branches formed below. There is a distinct superior-subordinate relationship. Figure 1 is an example of a hierarchical database structure based upon Date's supplier, parts, and shipments database (10:64). The data base is a forest of trees, each of which has a root node record. Below the root record are subordinate record nodes, each of which, in turn, owns one or more other nodes (perhaps none). Each node in the tree, except the root, has a single owner. Each of the records in the tree contains a collection of fields. Each field contains a single value. Because of the structure, data must often be replicated in several different locations within a hierarchical database. According to Korth and Silberschatz, this presents two major drawbacks: (1) data inconsistency may result when updating takes place, and (2) waste of space is unavoidable (20:144).

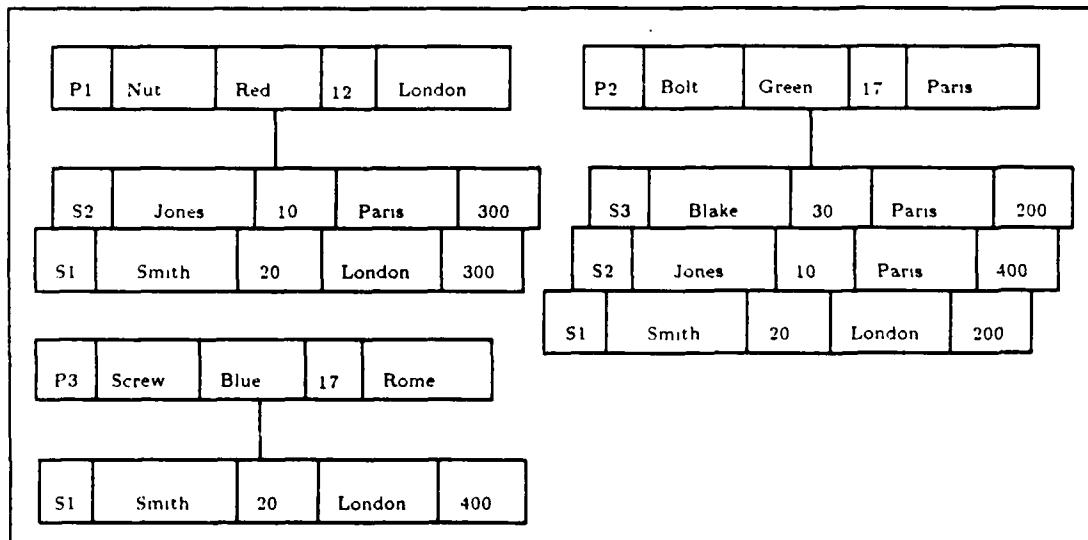


Figure 1. Typical Hierarchical Database

As database theory continued to develop, some of the problems inherent in the hierarchical

model were circumvented by the more sophisticated network model. Like the hierarchical model, a network database consists of a collection of records connected via links. Unlike the hierarchical model, the network model allows arbitrary graphs as opposed to trees. Thus, each node may have several owners and may, in turn, own any number of other records. The network model provides a mechanism by which a field can have a set of values. It also reduces the amount of replicated data inherent in the hierarchical model. Figure 2, which is borrowed substantially from Date, shows a typical network database (10:64).

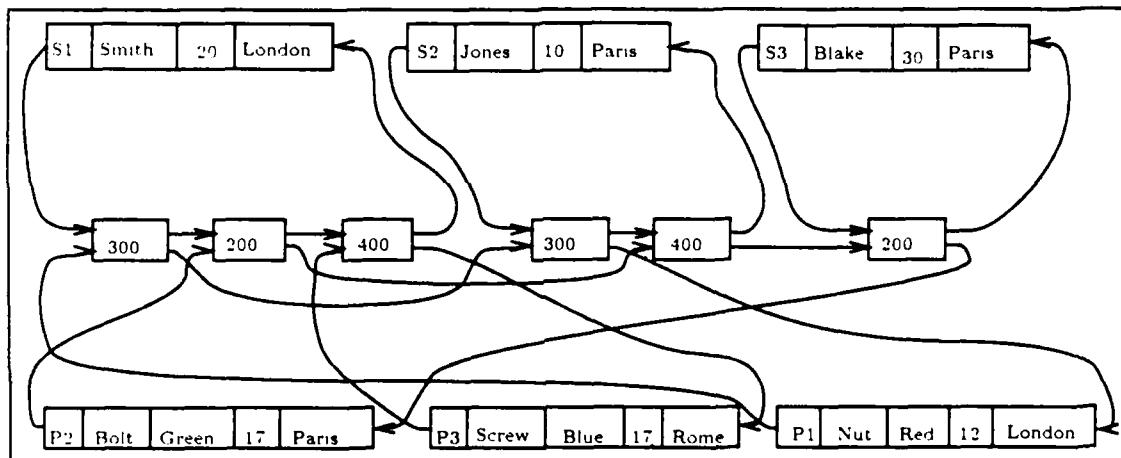


Figure 2. Typical Network Database

The Database Task Group (DBTG) of the Conference on Data Systems Languages group (CODASYL), which had set the standard for the COBOL language, studied a number of these network-based DBMS in the late 1960s. This study resulted in the first database standard specification, the so-called DBTG network model.

Research on database systems continued. E. F. Codd, at the IBM Research Laboratory in San Jose, introduced the relational model in his 1970 paper (7). A relational database consists of a collection of tables (relations), each having a unique name. Each table has a number of columns (attributes), which also have a unique name. The primary assumption of Codd's relational model is

that all attributes in a relation can only have atomic values, i.e., cannot be decomposed. A relation which only has atomic valued attributes is said to be in first normal form (1NF). Codd's paper includes a rigorous mathematical treatment of the subject. Additionally, his model provides the mechanism for separating the programs from the machine representation and organization of data (one of the big problems associated with both the hierarchical and network database models). A set of relations from Date's sample database, is shown in Figure 3 (10:64). According to Stonebraker, Codd's paper "started a heated controversy in all ACM SIGFIDET (now SIGMOD) meetings from 1971 onward between two collections of people..." (36:1). The previously mentioned CODASYL group was pushing the DBTG network model (their recently defined database standard), while Codd and academic researchers were pushing the relational model.

supplier			
s#	sname	status	city
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris

part				
p#	pname	color	wgt	city
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome

sup-part		
s#	p#	qty
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S3	P2	200

Figure 3. Typical Relational Database

Two influential prototype database systems based on the relational model were developed and subsequently commercialized. These two systems, "helped shape a fair amount of the history that followed" (36:2). The IBM Research Laboratory at San Jose built System R, and the University of California at Berkeley built Ingres. Ingres was eventually commercialized by several companies, including Relational Technology (now Ingres Corporation). System R was also commercialized by several companies including Oracle.

By and large, the relational model is the de facto database standard. However, research in the database arena has continued, and several other models have been proposed. One such model, which has found significant use during the design phase of databases, is Chen's entity-relationship model (6). Basically the E-R model extends the relational model with the concepts of entities, which are represented by rectangles; attributes, which are represented by ellipses; relationships, which are represented by diamonds; and the links between them, which are represented by lines. An example E-R diagram based upon Date's database is shown in Figure 4. There do not seem to be any commercial database systems that use the entity-relationship model as their underlying data model. Nonetheless, the E-R model has obvious uses, in particular for logical database design². In fact, many commercial relational database design tools require the database administrator to express data using the E-R model. Stonebraker explains why the E-R model never really took hold. "The relational model was dramatically and obviously better than the older hierarchical and network models... The E-R model, on the other hand was not seen to be dramatically better than the relational model" (36:369).

Stonebraker gives an overview of some other database approaches, for example; the functional model attempts to view the database as a collection of functions; the semantic data model is an attempt to deal with what Stonebraker calls the "semantic poverty" of the relational model (36). Another increasingly popular approach is the so called, object-oriented approach³. The

²The IDEF₀ data model in this thesis is derived via an entity-relationship analysis.

³The Exodus DBMS used to implement the nested-relational IDEF₀ database is an object oriented system.

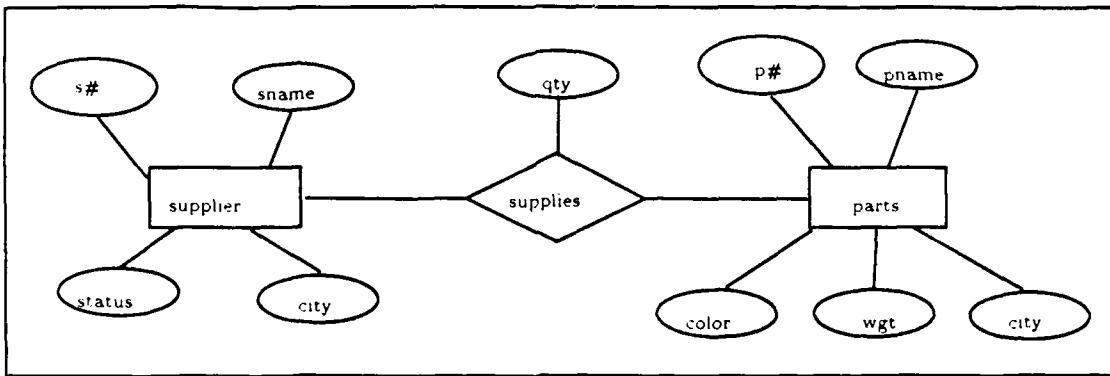


Figure 4. Typical Entity-Relationship Diagram

exact meaning of object-oriented varies from person to person. Bancilhon describes the main characteristics of an object-oriented system which should, in turn, be manifested in any DBMS that chooses to be called object-oriented. These characteristics include encapsulation, object identity, types and classes, inheritance, overriding and late binding, and degrees of freedom (2:152).

Another important database model is the nested-relational model, which is basically an extension of Codd's relational model. The extension allows for attributes within a relation to be multi-valued or even relation-valued, i.e., the 1NF assumption is relaxed. Perhaps the earliest work in this area was done by Makinouchi, who considered set valued attributes (22). Thomas and Fischer subsequently extended this concept to include relation-valued attributes (38). Roth, Korth, and Batory proposed extensions to the SQL query language (SQL/NF) so that it could deal with these non-first normal form ($\neg 1NF$) relations (33). The nested model, while retaining Codd's traditional operators, also has two new operators, nest, and unnest. These operators are best explained by way of the following example from Mankus:

... suppose a relation r is defined on some scheme R , with attributes A , B , and C . This may be denoted as $R = (ABC)$. If the attributes B and C are then nested under one attribute, thus giving us a relation-valued attribute, the scheme may now be shown as $R' = (AD)$, $D = (BC)$, where B and C are nested under the D attribute of R' . By unnesting R' with the unnest operator, the scheme $R = (ABC)$ is returned. (23)

One advantage of the nested model is that it can deal with complex, hierarchically structured objects wherein an object is composed of lower level subobjects. A good example of this is seen in a personal computer. The object "computer" is made up of subobjects such as "monitor", and "disk drive." Each of these subobjects, in turn may be comprised of subobjects. For example, "monitor" is comprised of subobjects such as "circuit board," which, in turn, is made from "integrated circuits," "resistors," "capacitors," etc. Each subobject is dependent upon its parent object. If the parent is deleted, so are all its subobjects. Although there do not seem to be any commercial databases based upon the nested relational model, there are prototype research systems such as that built by Mankus (23).

The current situation is that relational database systems work quite well for business data-processing applications. However, if you "stray from data that looks like the SUPPLIER-PARTS-SUPPLY example popularized by Codd or the EMP-DEPT examples also in widespread use, relational systems tend to run into trouble very quickly" (36:477). In nonbusiness application areas the opportunity to stray is rampant. For example; CAD applications need to store two-dimensional and three-dimensional objects in some type of database; considerable research has been expended on trying to integrate database management systems with artificial intelligence systems; Rubenstein describes the design of a database system for musical information (34). It is almost certain that next-generation CASE tools will put software specifications, definition of forms, reports, graphs, and even source code in a database. In all these areas, current relational systems tend to work poorly; user queries are difficult to construct and they execute slowly. As a result, developers of these kinds of applications generally ignore relational technology. Some other database approach might prove useful. Particularly in a data intensive area such as CASE tools.

Problem Statement

The purpose of this thesis investigation is to analyze the data requirements of the IDEF₀

structured analysis language; develop an abstract data model of the language; implement this data model within an Ingres-based relational DBMS; implement the model within an Exodus-based nested-relational DBMS; compare the two implementations in terms of query complexity, size of the database, and speed of query execution; and, based upon the comparison, determine if the nested-relational implementation of the model is more appropriate for the IDEF₀ application.

Plan of Attack

The plan of attack is to analyze the data requirements of the IDEF₀ language and develop an abstract data model. After the data model is complete, it is mapped to a set of relations and implemented in the Ingres relational DBMS. An example database instance, and a series of queries is developed to extract data from the relational implementation. The relational model is then mapped into a nested-relational representation and implemented within the Exodus-based nested-relational DBMS. An example database instance containing the same information as the relational instance is developed, as well as a series of queries to extract data from the nested-relational version. Finally, comparisons between the two implementations are generated. These comparisons include complexity of queries, speed of query execution, and size of the database.

Development of an Abstract Data Model. In order to facilitate development of the relational and nested-relational DBMS implementation of the IDEF₀ language, an abstract data model of the language is constructed. This abstract data model consists of two parts, the essential data model, and the drawing data model. The former captures only the essence of the analysis language in terms of activities and data elements, whereas the latter captures only those portions of IDEF₀ which are strictly graphical constructs. This approach allows future tools to extract analysis data without having to deal with the fact that the analysis language was IDEF₀. The concept is illustrated in Figure 5.

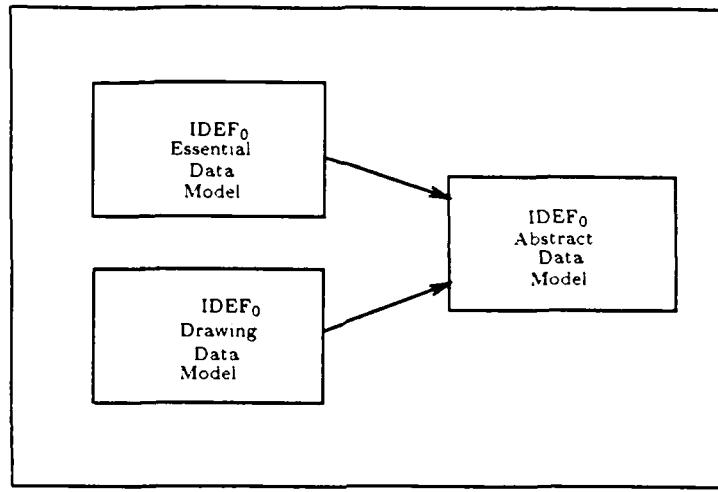


Figure 5. IDEF₀ Abstract Data Model

Essential Data Model. The IDEF₀ essential data model captures those portions of the language which represent the semantics (relative to a human interpretation) of a particular analysis. This includes, for example, activities and their children, as well as data elements. It does not include, for example, the location of the boxes or arrows which graphically represent the activities and data elements. As mentioned earlier, this approach allows future tools to extract analysis information without having to deal with the IDEF₀ graphical representation explicitly, i.e., without having to "walk through" the drawing.

The entity-relationship model is used to analyze the essential data model since it allows for easy mapping into a relational design. Furthermore, one of the advantages of the entity-relationship approach, as explained earlier, is that it retains many of the semantics of the actual data being modeled.

Drawing Data Model. The drawing data model represents the actual graphical constructs, e.g., boxes, arrows, etc., used to represent the particular IDEF₀ analysis. This data is used to draw an IDEF₀ model. It contains such information as the location of boxes, the line segments

which graphically represent a given data element, etc. For the same reasons as before, an E-R analysis is used to derive the model.

Mapping of Abstract Data Model. The mapping of an E-R model into the corresponding relations is relatively straight forward. An example of a mapping approach is given in Chapter 2 of Korth and Silberschatz (20:21). After the mapping into relations is accomplished and tuned via stepwise refinement, it is a simple task to implement the design within the Ingres DBMS. The mapping into a nested-relational model is a different matter. There are paradigms for mapping a scheme into a nested-relational design given a universal relation, its functional dependencies, and multivalued dependencies (31). Unfortunately, there is no easy way to determine multivalued dependencies, particularly if the data model is developed using classical methods such as E-R. Thus, the development of the nested-relational design is essentially still an art form which relies on the skill of the analyst. Following the nested-relational mapping is the implementation within the Exodus DBMS.

Example Database Instance. A two-level IDEF₀ analysis is used as the basis for the example database instances. This instance is loaded into the relational implementation, and the nested relational implementation. Identical instances in each version allow for a somewhat normalized comparison.

Development of Queries. Typical queries to extract data from the example database are developed. For the relational version, the query language is SQL. Queries to extract the identical data from the nested-relational implementation are developed. Although the query language for Mankus's nested-relational DBMS is based upon Colby algebra, the ultimate goal is to build a query language front end based upon Roth's SQL/NF (8) (33). Accordingly, the nested-relational queries are written in SQL/NF and translated into Colby algebra. The justification for writing the

queries in SQL/NF pertains to the query complexity comparisons between the nested-relational version and relational version.

Comparison. The comparison is a somewhat difficult issue. One area that could provide useful data is in the area of query complexity. In order to determine which query is "more" complex, complexity is quantified in terms of query language constructs. The queries associated with the relational and nested-relational implementations are compared based on these complexity measures.

The comparison of the size of the database files seems to be rather straight forward. Unfortunately, a simple comparison of file size is not necessarily meaningful. Is the smaller size due to an intrinsic property of the data model or is it due to the skill of the database programmer in choosing a data structure? The best that can be done with this approach is to build several database instances and attempt to draw some qualified conclusions. On the other hand, if a relational example database instance and a nested-relational example database instance (containing the identical data) are compared byte by byte, on paper, then the logical size of each can be determined. This provides a worst case comparison, since internal representations tend to compress data.

While it is rather easy to simply compare the running time of the two implementations for various queries, the numbers are again not necessarily meaningful. Is the faster running time of one model due to an intrinsic property of the model or is it due to a more skillful programming effort? Does the Ingres version run faster because there are fewer people logged in? Does Exodus run faster because it is on a Sun workstation? The best that can be done with such an approach is to examine a number of different queries and attempt to draw some qualified conclusions. A more reasonable approach is to consider query speed in terms of number of disk accesses (assuming a disk based DBMS approach), or an order-of analysis on programs which run the embedded query language queries (assuming a memory based DBMS). This latter approach obviously depends upon

the size of a given IDEF₀ analysis. It may not be possible to load the complete set of data for a given project into memory at one time.

Scope and Limitations

Scope. The thesis effort covers four specific areas as indicated below:

1. Development of an abstract data model for the IDEF₀ language.
2. Design and implementation of an Ingres-based relational database to capture the IDEF₀ data.
3. Design and implementation of an Exodus-based nested-relational database to capture the IDEF₀ data.
4. Comparison of the two DBMS implementations to determine the benefits, if any, associated with the use of a nested-relational DBMS for such applications.

Limitations. The development of the abstract data model, and the nested-relational implementation of this model are the primary areas of emphasis. The ability to use the abstract data model in other on-going thesis efforts relative to IDEF₀ is of primary concern. In particular, Smith's Ada implementation of SAtool uses the data model, as does Austin's implementation of a Structured Analysis Tool Interface to the Strategic Defense Initiative Architecture Dataflow Modeling Technique (35) (1). Another area of emphasis is Kirkpatrick's dissertation efforts relative to nested-relational DBMS. The nested-relational IDEF₀ implementation depends upon Mankus's development of a nested-relational DBMS (23). If the nested-relational DBMS is not robust enough to implement the nested-relational design, a "paper model" will be constructed and used as the basis for comparison. Finally, the ability to map the abstract data model into the existing AFIT software development environment is of concern (16:8).

Sequence of Presentation

This thesis consists of five chapters. An overview of the IDEF₀ language, and the Exodus extensible DBMS, as well as a literature review of DBMS as each applies to CASE tool data is presented in Chapter II. The design of the IDEF₀ abstract data model and the corresponding relational and nested-relational DBMS implementations are presented in Chapter III. Chapter IV summarizes and compares the IDEF₀ implementation within a relational and nested-relational DBMS. Finally, Chapter V presents the conclusions of this research effort and includes recommendations on further research in this area.

II. LITERATURE REVIEW

Introduction

The purpose of this investigation is to develop an abstract data model of the IDEF₀ language, implement the data model within a relational and nested-relational DBMS, and determine if there are benefits associated with the nested-relational implementation of the model.

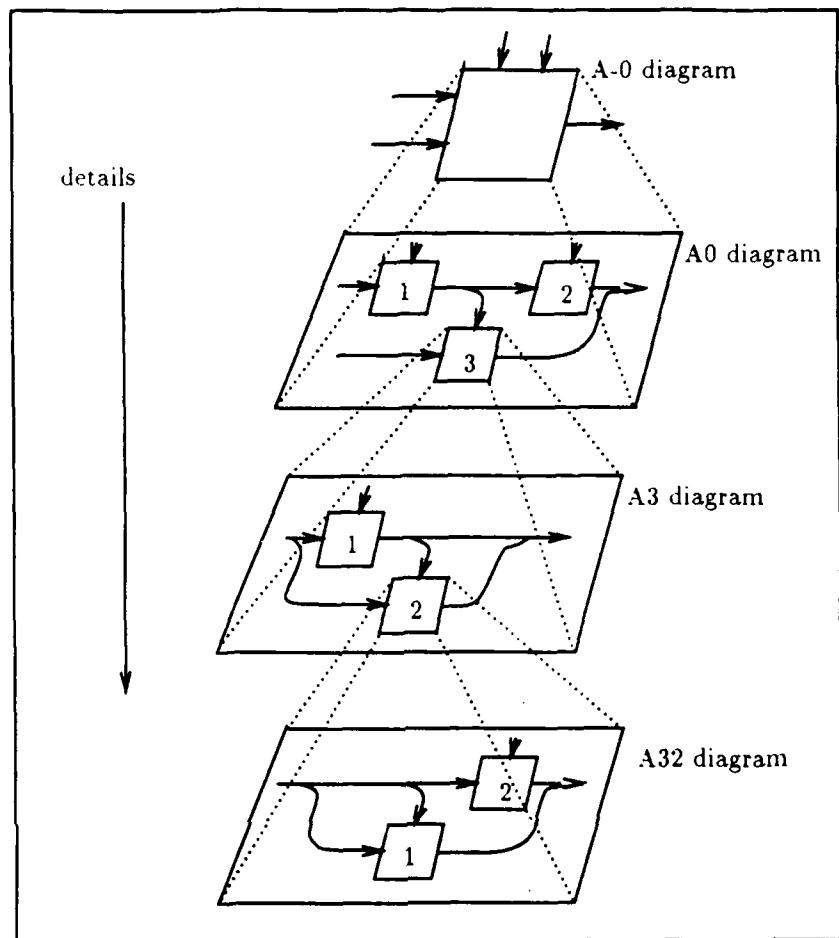
Since the particular language being implemented is IDEF₀, and the implementation DBMS for the nested-relational model is Exodus, a brief overview of each is presented. The underlying issue of course, is to determine if the mapping of CASE tool data into a DBMS is worthwhile. Accordingly, a literature review of CASE tools and their connection with DBMS is conducted to gain some insight into the problem.

Overview of IDEF₀

IDEF₀ is a graphical language which, among other things can be used during the analysis phase of software development. In order to discuss IDEF₀, it is necessary to also discuss Structured Analysis (SA). The following paragraphs give a brief overview of both SA and IDEF₀.

In his 1976 paper, Douglas Ross introduced Structured Analysis as a generalized language, which allows a complex idea to be represented in a hierarchical, top-down representation (30). According to Ross, "The human mind can accommodate any amount of complexity as long as it is presented in easy-to-grasp chunks that are structured together to make the whole" (30:17). SA combines graphic features such as lines and boxes with standard written language to create the SA model. Figure 6 illustrates the basic idea behind this structured decomposition. At each level, only the details essential for that level are given. Further details are exposed by moving down in the hierarchy.

SA provides for two kinds of decomposition, an activity decomposition, and a data decomposition. In the activity decomposition, activities (verbs) are represented by rectangular boxes, and



Based on (30:18)

Figure 6. Structured Decomposition

data (nouns) are represented by arrows flowing into and out of the boxes. In the data decomposition, boxes represent data, and arrows represent activities operating on the data in the boxes. An example of an activity decomposition is shown in the following two figures. Figure 7 represents the overall context of the system being analyzed (the so called "A minus zero" diagram). Figure 8 represents the first level decomposition (the "A zero" diagram). In a real analysis, the A0 diagram would be further decomposed to whatever level was necessary to ensure an unambiguous interpretation of the system requirements. Marca and McGowan have written an excellent book which

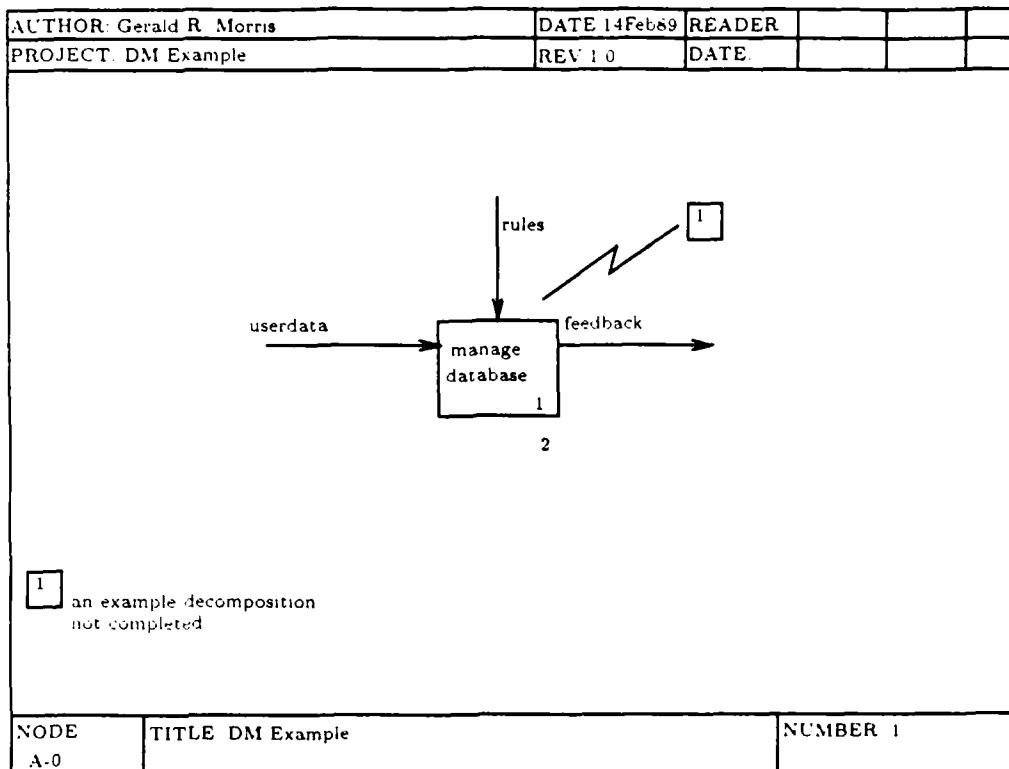


Figure 7. A-0 Diagram

describes SADT¹ and provides numerous workshop-style examples with which users can develop a flavor for the language (24).

A full implementation of Ross's SA includes 40 different language features, and the dual decomposition (30:20). But the United States Air Force Program for Integrated Computer Aided Manufacturing (ICAM), which is directed towards increasing manufacturing productivity via computer technology, defines a subset of Ross's Structured Analysis language called ICAM Definition Method Zero, or just IDEF₀ (25). This functional modeling language eliminates some of the more esoteric features of Ross's language, as well as the data decomposition. Appendix B shows the features of the IDEF₀ language.

According to the IDEF₀ manual

¹ Structured Analysis and Design Technique (SADT) is SofTech's name for SA.

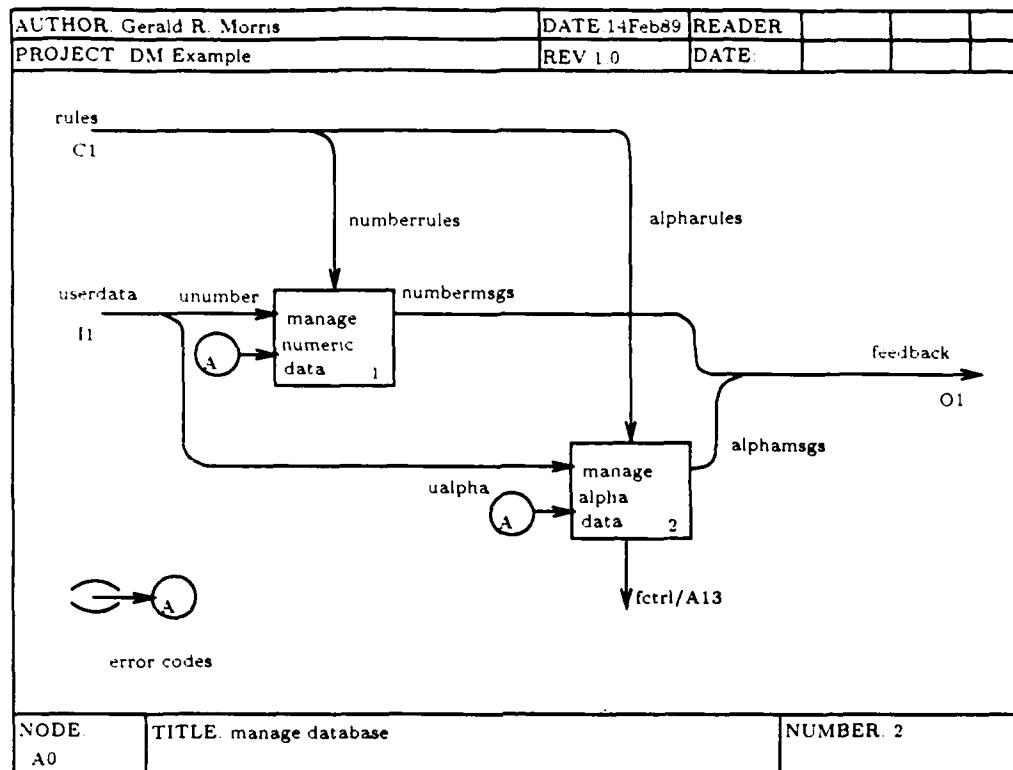


Figure 8. A0 Diagram

The ICAM program approach is to develop structured methods for applying computer technology to manufacturing and to use those methods to better understand how best to improve manufacturing productivity. . . . IDEF₀ is used to produce a **function** model which is a structured representation of the functions of a manufacturing system or environment, and of the information and objects which interrelate those functions. (25:1-1)

One of the problems with both SA and IDEF₀ is that there does not appear to be a formal model of the language. In addition to "blueprint-like graphics," SA and IDEF₀ call for the use of natural language (30). The use of such natural language, by definition, introduces ambiguity in the overall IDEF₀ language. In addition, certain graphical features of IDEF₀ allow for ambiguous models to be constructed. In short, IDEF₀ is not a rigorous language. Although the original intent of the IDEF₀ language was to provide a structured approach for computerizing manufacturing processes, it is also the language being used by the Strategic Defense Initiative Organization (SDIO) to help understand the requirements for the so called "star wars" defense. The language also provides

an analysis methodology for the requirements-phase of a development effort. It is precisely this use of IDEF₀ as a software requirements methodology which was the motivation for developing the data model presented in this thesis. In addition, the model helps mitigate some of the ambiguities inherent in IDEF₀.

AFIT and IDEF₀

The Air Force Institute of Technology (AFIT) Department of Electrical and Computer Engineering has promulgated a set of system development guidelines and standards which encourage consistency throughout all phases of hardware and software systems development (16). As part of this standard, and in conjunction with ongoing efforts to utilize computer aided software engineering (CASE) in the software engineering curriculum, the department selected IDEF₀ as the language of choice for performing systems analyses. The IDEF₀ language was extended to include a data dictionary, which is AFIT's implementation of and improvement over the glossary called for by IDEF₀. It provides not only the glossary, but also a more syntactical representation of some of the ambiguous features of the language.

Several past efforts have produced CASE tools to assist AFIT students during software development. In particular, Johnson developed a tool called SAtool, based upon the IDEF₀ language, which allows the software engineer to perform an analysis of the software requirements (19). SAtool, which runs on a Sun workstation, is a graphics based editor which allows the analyst to draw the diagrams and enter portions of the data dictionary for the requirements analysis phase of software development. The remaining elements of the data dictionary are automatically derived from the diagram. The user can generate a printout of the SA diagram, a so-called *facing-page text* printout, and a hard-copy printout of the data dictionary. The analysis results can be saved in a standard data file for uploading into a common database. The tool also saves the graphical drawing informa-

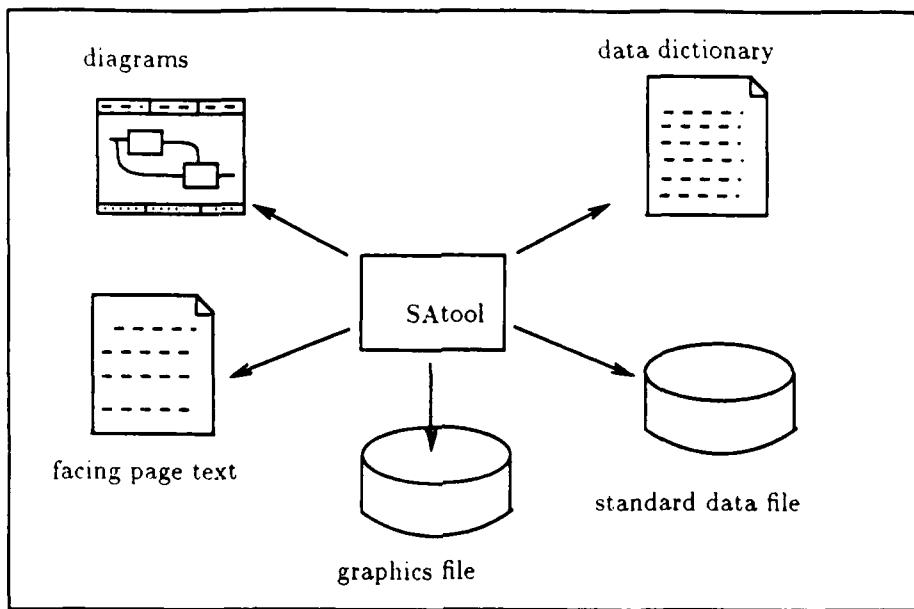


Figure 9. SAtool Products

tion so the user can recall the diagram for editing. Figure 9 illustrates some of the SAtool products. and Appendix C contains some typical examples of these products.

The standard data file generated by SAtool can be uploaded into AFIT's common database. According to Connally, the goal of the common database system is to

provide an integrated system in which a designer could sit down at a workstation, download the necessary data from a central database, work on a portion of the design, and when finished, upload the data back to the database. (9:2)

Overview of Exodus

There are several ongoing research efforts within the database arena which are attempting to deal with non-traditional database applications. There seem to be two general approaches; build a database that has all the capabilities needed for non-traditional applications; or build an extensible database that can be tailored to the needs of a specific application. One such project, which falls

into the second category, is the University of Wisconsin's Extensible Object-oriented Database System (Exodus).

Basically, Exodus can be perceived of as a toolkit which allows an application-specific database to be constructed on top of an existing set of kernel facilities. Carey provides the following abstract description of Exodus:

The goal of this project is to facilitate the fast development of high-performance, application-specific database systems. Exodus provides certain kernel facilities, including a versatile storage manager. In addition, it provides an architectural framework for building application-specific database systems; powerful tools to help automate the generation of such systems, including a rule-based optimizer generator and a persistent programming language; and libraries of generic software components (e.g., access methods) that are likely to be useful for many application domains. (4:1)

The Exodus architecture provides the following tools to be used in building a database:

- (1) The Storage Manager.
- (2) The E programming language and its compiler.
- (3) A library of type-independent Access and Operator Methods.
- (4) A rule-based Query Optimizer Generator.
- (5) Tools for constructing query language front ends. (4:3)

It is Exodus's toolkit quality which makes it ideal for implementing a nested-relational DBMS to manage the data associated with IDEF₀. Although the Wisconsin researchers have already built a relational DBMS, called Exrel, using the Exodus toolkit, it is not robust enough to handle the relational implementation of IDEF₀ (3). The nested DBMS built by Mankus is used as the implementation media for the nested-relational version of the IDEF₀ database (23).

CASE Tools and DBMS

A data base is invariably at the center of any kit of CASE tools. As experience with such data bases accumulates, CASE vendors are finding more ways to use stored information to aid

in the software-development process. Because it is a repository for the specifications developed during the analysis phase, the data base can act as a bridge to the design phase, making specific information on system functions available while the design is being partitioned and detailed. As the storage place for the data elements and code modules that make up the final design, the data base becomes the source for reconfiguring the software after design changes and for reuse of code, as well as specification and design elements. The data base is also the source from which information for documentation, project management and software testing is drawn.

An Analysis Scenario - The Need for a DB.MS. The following scenario, associated with the development of an analysis via AFIT's SAtool, demonstrates an initial input sequence, modification sequence, and several queries. The section closes with a discussion as to why a database management system might be useful for storing the data.

Creating SA Diagrams. The following is a brief description of a typical SAtool session. Using the graphics based SAtool editor, the analyst creates the A-0 top-level diagram shown in Figure 7. The graphics frame includes attributes such as author, date, revision number, project name, the title of the diagram, diagram number, and node number. For the single activity box on the diagram, there are several attributes. Examples include activity name; the input, output, control, and mechanism arrows attached to the activity; a description of the activity; and an activity number. Each of the data arrows also includes several attributes. For example, data name, description, data type, range, value, and composition (data arrows nested inside another data arrow). Typically, the analyst does not yet know the composition of "high level" arrows; more often than not, this is determined at the next lower level. In any case, after drawing the boxes and arrows and entering all of the known data associated with the A-0 diagram, the analyst then saves the drawing data in a flat file and the data dictionary data in another flat file.

A new drawing is then started which decomposes the parent activity box to expose the details in the next level as shown in Figure 8. Here the analyst essentially starts over again. He draws

all the boxes, all the arrows, enters all the data dictionary items, and saves the result to another pair of flat files. Notice that much of the information on the A0 diagram is the same as on the A-0 diagram.

Modifying an SA Diagram. Suppose the analysis documents need to be modified. For example, a requirement to handle dates gets added. The analyst must open the file for the A-0 diagram and edit the data dictionary for the three data arrows **userdata**, **rules**, and **feedback** to include **update**, **daterules**, and **datemsgs** respectively, in the composition fields. He would enter the new revision number and date and then save the results (if the old version was required as history, its files would have to be saved elsewhere). The analyst would then open the A0 diagram, add a new activity box **manage date data**, and add the new data arrows between **userdata**, **rules**, **feedback** and this new activity box. He would enter all the data dictionary requirements for the new arrows and activity box, and save the results.

Querying an SA Diagram. Obviously there will be a number of predetermined "standard" queries associated with the SA diagram. For example, the queries needed to extract the drawing data, or the queries needed to move from one level to another, or the queries needed to determine the data content of a parent arrow. Certainly, even without the use of a DBMS, it is possible to write these standard queries in some high-level language and make them available to the tool user via the standard tool interface mechanism. However, ad hoc queries are another matter. Let's look at several queries that might be typical for an SA analysis (these queries are based on my own personal experience with software analyses in general, and with SA in particular). Suppose the analyst is interested in determining all of the activity boxes which are touched by the **userdata** data arrow. This is currently done by painstakingly examining each diagram and manually tracing the arrows. There is no effective way of accomplishing this via the current automated tool. Suppose the analyst is interested in all the primitive (lowest level) activities within a certain activity box. Once again this must be done by manually looking at each of the subordinate diagrams and

extracting the activity boxes which have no "children." Suppose a certain analyst on the team gets promoted. His replacement, Mary, needs to know which diagrams now belong to her. This would currently be done by manually examining each diagram to determine the author. Finally, suppose the project manager needs to determine which diagrams were modified after a certain date. Once again this would be done by manually examining each and every diagram.

Why a DBMS Would be Useful. The previous sections consider a typical analysis scenario in terms of creating, modifying, and querying the analysis products. Some of the problems associated with the current, non-DBMS implementation of SAtool are illustrated. This section suggests that a DBMS might be useful in providing a solution to each of the problems.

Consider the initial creation of the A0 diagram. Clearly, much of the information needed on the diagram is already available from the previously generated A-0 diagram. For example; the project name, title, etc., should be propagated down to subordinate levels. Unfortunately, the current implementation of SAtool requires the user to enter all of this information at each level. As a result, there is a significant amount of redundant data stored for each level of the diagram. Say that an analysis consists of N levels, and at each level (except the A-0), there are k boxes. Assume that the only redundant data is the project name. It gets stored once at the A-0 level, k^0 times at level 0 (A0), k^1 times at the level 1 (nodes A1 through Ak), k^2 times at level 2 (nodes A11 through Akk), k^3 times at level 3 (nodes A111 through Akkk), ..., k^N times at level N. Adding this all up we see that the same piece of data has been redundantly stored R_N times, where

$$R_N = 1 + \sum_{i=0}^N k^i = 1 + \frac{1 - k^{N+1}}{1 - k}.$$

The formula above is conservative; SAtool actually saves project name for each box and for each data arrow! The simple 2-level analysis depicted in the two SAtool figures results in a flat data

file which contains the project name "DM Example" 14 times. If this information were available in a database, it could simply be referenced by the subordinate level diagram.

Aside from the redundant storage issue, there is the high potential for incorrect data entry. All of the input is done manually at each level; no consistency checking is done. It is possible to create diagrams that absolutely do not match at the various levels. A DBMS, which typically includes consistency validation routines, could help mitigate this problem.

The modification scenario presented above could be significantly simplified by the use of an appropriate DBMS. When the analyst decomposes a pipe data element into its constituent data elements, their names, et al, would be automatically propagated upward into the higher level diagrams. As already mentioned, it is currently possible to decompose an arrow (data item) at a lower level and then not update its "parent" arrow. The fact is that an IDEF₀ analysis, being a top-down approach, gradually exposes more detail as one moves down the hierarchy. Generally speaking, the analyst does not even know the composition of a given "high level" data arrow until the lower level diagrams are drawn. Accordingly, the analysis typically goes something like this: draw the high level diagram and save it. Draw the lower level diagrams and save them. Reopen the high level diagram and add the correct composition to arrows. Clearly this redundant effort might be eliminated by the use of a DBMS.

The queries discussed above could also benefit from a DBMS. The drawing tool could use an embedded version of the DBMS query language to extract and draw the diagrams. The user could employ the query language of the database manager to extract information. Stored queries could be maintained for those queries which occur frequently; ad hoc queries could be constructed on the fly. Finally, the use of a DBMS would automatically provide for crash recovery and concurrency control for multiuser applications.

Mapping Difficulties. It is certainly not yet clear that the nested-relational DBMS is the answer to all the problems associated with mapping CASE tool data into a DBMS. At the risk of

appearing prejudiced against relational databases, I would like to illustrate some of the problems that arise when using a relational DBMS in a CASE tool data environment. For example, a design object may be physically too large to fit in a standard record, and if a single design object is represented by many records, the user must still be able to manipulate it as if it were a single unit. In addition, different users may want different views of the same object. One user might want to see a high-level data-flow view of the "Guide-Torpedo" system, whereas another user might want to see only the processes associated with "Guide-Torpedo," or perhaps only the main routine from the design phase data dictionary.

One of the biggest problems associated with mapping CASE data into a relational model is that the record oriented relational model forces the designer to oversimplify data structures to the extent that information about the database is lost. Logical entities must be broken up into many relations in order to "force fit" them into the relational model. This typically results in a large number of relations and tuples. In a design phase database, for example, the code of a module body typically involves perhaps 100 lines. Assume that each line is restricted to an 80-column format. Clearly we would need to map this into 100 tuples each having (among other fields) an 80 character string field. In order to retrieve the single logical entity called "code body," we would have to retrieve 100 tuples from the database and then (somehow) force these lines into a single text file. Figure 10 is an admittedly simplified E-R diagram of the SAtool data model. There are essentially two entities: activity boxes, and data arrows. Nonetheless, when we map this into a relational model, we end up with some 23 different relations as shown in Appendix D. A direct consequence of this mapping is that query processing is slowed down by virtue of the multiple joins required.

Another problem associated with mapping CASE data into a relational model is the length (time wise) of transactions. In a typical business application, a transaction only lasts a few moments. The user grabs the applicable tuples, immediately makes changes to them and writes them

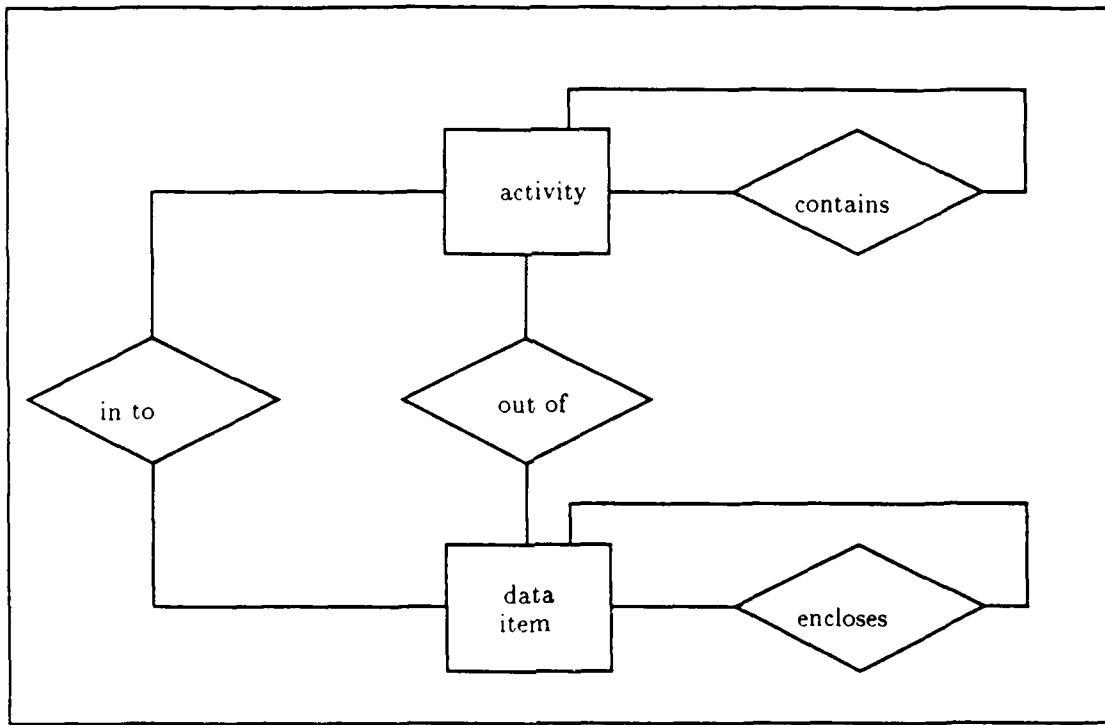


Figure 10. Highly Simplified E-R Diagram of SAtool Data Model

back to the database. In CASE tools however, the user checks out the appropriate diagram (tuples?), and then may spend hours if not days making changes before checking in the results. In a typical relational DBMS, this might seriously impact the crash recovery capability of the system.

Yet another problem when mapping CASE data into a relational model is that of keeping revisions (history). A typical software development effort requires the developer to keep all revisions in the database. Thus, when we "update" a logical entity, we might be adding a new feature to the entity, or we may be simply changing an existing feature, i.e., the meaning of the word "update" is now ambiguous. A similar problem exists with "delete;" say a certain revision, y, requires us to delete module, x. Clearly we want to keep the entities associated with module x in the database for revisions earlier than y, and "delete" them for revisions y and beyond.

As a result of these mapping problems (and perhaps others), existing database technology

forces many CASE vendors to defer to the file-based structures that early business applications used. The current generation of CASE tools, for the most part, are special-purpose systems in which a collection of files represent design objects. The primary drawbacks of this approach are the lack of data independence, the complexity of database administration, the high degree to which the data is tied to the machine representation and internal data structures, and the lack of fully general concurrency and recovery systems. These drawbacks are, in many respects, identical to those encountered by data-processing applications before database management systems came into widespread use.

Commercial Databases for CASE Tools. There are a few vendors who have developed databases tailored specifically towards the needs of CASE tools. Three such systems are briefly described in the following paragraphs.

CDD/Plus. The VAX Common Data Dictionary, *CDD/Plus*, developed by Digital Equipment Corporation (Marlboro, MA), is a distributed data dictionary which allows users access to software data definitions either centrally or locally across a network. It spans the entire software life cycle—from applications development and code generation, through production and systems implementation. It allows data extraction to simplify software documentation efforts, and allows ad hoc analysis and reporting (37).

The Developer. The Developer, available from Asyst Technologies, Montreal, Canada, lets the user construct simple diagrams consisting of rectangles, interconnecting lines and arrows, and text. In addition to storing items such as data elements and data-flow descriptions, a data base contains attribute information for each item, as well as associations between stored items. The user can decide not only which items to store, but also what attributes the items will have and what the rules will be for associations between the items (17).

vsDesigner. *vsDesigner*, from Visual Software, Santa Clara, CA, is a highly flexible database oriented specifically towards the CASE market. It is object-oriented and lets users define the objects and object attributes. Associations between objects can also be defined in the form of rules. Each object has multiple user-defined representations: graphics symbols, object descriptions, rules for using it with other objects, and attributes. During analysis, users can install links in specification objects that will later be connected to design objects. Source code can also be associated with each object, along with attributes such as execution times. With the help of a data base query language called *vsSQL*, users can make execution-time analysis of real-time software by walking through the branches of the software design and summing the object execution times (17).

Integration of CASE Tools. CASE tools are providing substantial productivity gains during the initial phases of software development, but CASE tools haven't generally been integrated with software coding, debugging or testing. This is largely due to the file based architectures discussed previously. Nonetheless, there are some ongoing attempts to alleviate this problem and provide data availability across all phases of the development effort. The following paragraphs provide insight into some of these efforts.

Standardized File Interchange Solution. One effort to solve the problem of CASE connectivity has been proposed by Cadre Technologies (Providence, RI), a leading CASE vendor. Their proposal has been submitted to the Electronic Design Interchange Format (EDIF) Technical Subcommittee, which makes recommendations on standards to the American National Standards Institute (ANSI). EDIF Review Process Committee members include a rather impressive list of well-known companies including; Advanced Technologies Applications, the Aerospace Corp., Apollo Computer, Applied Microsystems, Atherton Technology, Cadre Technologies, Deere and Co., E-Systems, Expertware, Hewlett-Packard, Index Technology, Integrated Data Ltd., I-Logics, Mark V Ltd., ProMod, Ready Systems, Sage Software, and Textronix. The benefits of an EDIF standard, according to Vizard, will include more competition among CASE tool vendors, greater

compatibility among products, increased specialization for particular CASE tools, better benchmark comparisons, and greater integration between contractors and subcontractors, irrespective of the CASE tools they are using (39). Basically, this adaptation of the EDIF standard will allow each tool vendor to exchange data through a standardized interchange file format. Unfortunately, the standard has not yet been approved by all parties.

Microprocessor System Development Solution. To provide CASE tool integration within the embedded system development arena, Microcase (Beaverton, OR) is teaming up with Cadre Technologies. Microcase manufactures the Software Analysis Workstation, an IBM PC-based system that provides performance optimization and verification for embedded software. The two companies are now developing an interface that will let data from the Software Analysis Workstation be captured by Cadre's data base. Microcase also sells compilers, debuggers, and in-circuit emulators. The partnership with Cadre gives Microcase the opportunity to build a complete microprocessor development solution, from front-end CASE tools to coding, hardware-software integration, and test (14).

VAX/CDD-Excelerator Solution. There are other ongoing efforts to integrate CASE tools across the software life cycle. Index Technology has developed a link integrating its systems analysis and design software, *Excelerator/IS*, with the VAX Common Data Dictionary, *CDD/Plus*. *Excelerator/IS* is the preeminent CASE system for commercial systems analysis and design. *CDD/Plus* was discussed in the previous section (37).

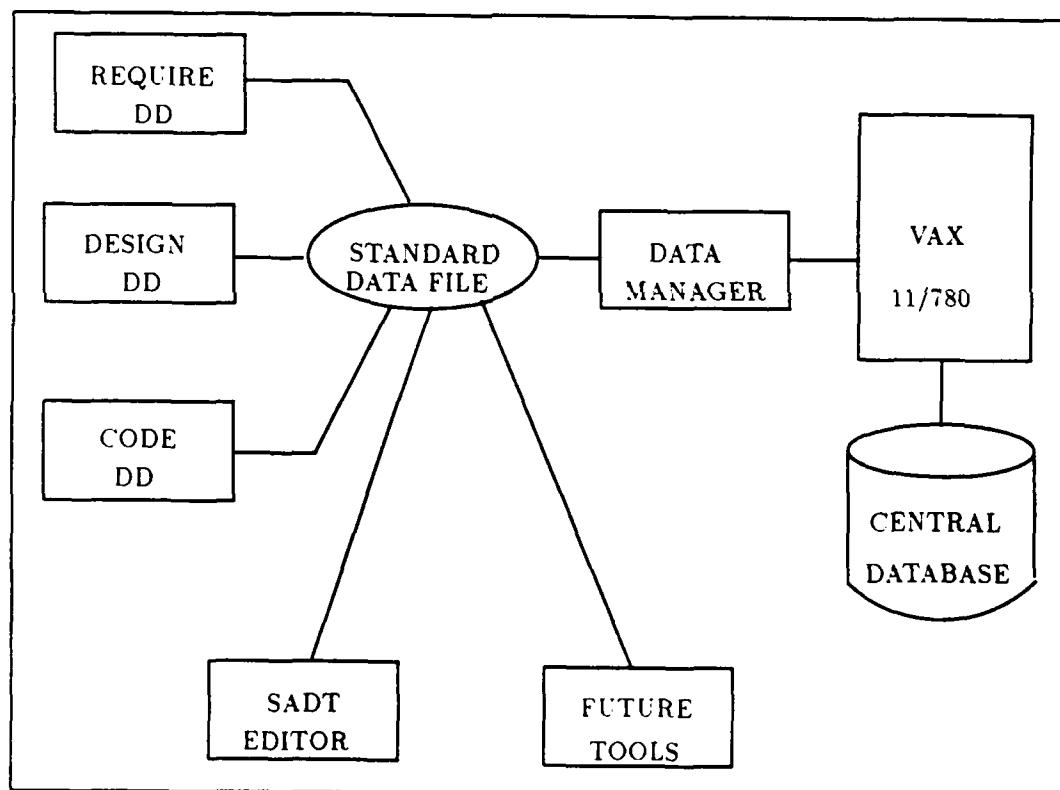
A Generic DBMS Solution. Goering discusses a standardization effort undertaken by Atherton Technology (Sunnyvale, CA) and Digital Equipment Corporation (Marlboro, MA) (15:28). Their solution, currently called ATIS, involves defining a way for tools to link into a consistent data-management system. The ultimate result will be a public-domain, nonproprietary database standard. Companies involved include Atherton Technology, Digital Equipment Corporation, Apollo

Computer, Cadre Technologies, Ford Aerospace, Hewlett-Packard, IBM, Index Technology, Interactive Development Environments, Interleaf, RCA, Rockwell, and Sun Microsystems. This effort attempts to accomplish the same goal as the proposed EDIF standard previously discussed. However, the ATIS effort allows a so-called "deeper level of integration" by defining a common way of managing multivendor data, as opposed to the simple data interchange format proposed by Cadre's EDIF solution. ATIS defines an object oriented methodology that provides an interface between CASE tools and data-management services, and it establishes models for such procedures as version control, security and access control, and transaction control. According to Goering,

The tool interface is based upon a predefined, single-inheritance hierarchy of data types. As is typical in object-oriented programming, each type has associated messages (such as open, merge and check-in), methods (pieces of code that implement messages) and properties (such as child and parent). To add a new tool, the tool integrator can either use existing types or add a new subtype. A new subtype can inherit existing methods and messages, or new methods and messages can be added. A tool that's designed with ATIS in mind can be integrated more efficiently than an existing tool. By supporting the predefined types, the tool can exchange data more efficiently and avoid duplicating storage. ATIS doesn't mandate a specific system for implementing a management service, nor does it dictate a specific type of data base. It does however, set forth some conceptual models that describe the execution of messages. Under the current version control model, for example, two users can open a file concurrently and update it locally. After the files are checked in, the management system merges them into a new version. ATIS provides several types of models including those for security and access control, which determines who has clearance to access data; those for naming services, which establish a file naming methodology; and those for transaction control, which guarantee data base consistency during concurrent multiuser access. The ATIS group also plans to address correspondence control, which establishes relationships between objects in the database. Another area that will be considered is data access and communication across a network. (15:28)

AFIT's Common Database Interface Solution. Many of the same problems being addressed in the commercial marketplace are also being addressed by the Software Engineering Laboratory here at AFIT. In his recent thesis effort, Connally designed and built an interface into an Ingres database which captures information from the analysis, design, and coding phases of the software life cycle (9:9). Figure 11 illustrates Connally's basic idea. The system allows the requirements phase data dictionary editor, the design phase data dictionary editor, the coding phase data

dictionary editor, and the SAtool editor to communicate, via a standard data file, with a centralized database. In theory, his system provides an ideal solution. However, several issues complicate things. First, the analysis tool, SAtool, runs on a Sun workstation and the Data Manager runs on a VAX 11/780. This requires network transfer of Connally's standard data file. Also, the data is being captured in a relational database. This requires a large number of relations as discussed earlier. As a result, the transfer of data to and from the Ingres database takes 10 - 15 minutes per session. In short, we have less than an ideal solution. Appendix E includes a typical Data Manager session.



(9:28)

Figure 11. Common Database System Structure

Summary

This chapter presents a brief overview of the IDEF₀ language, and the Exodus extensible DBMS. It also investigates the use of database management systems to support computer aided software engineering (CASE) tools. The connection (or lack of connection) between CASE tools and DBMS is considered. Since many commercial CASE tools do not seem to use relational DBMS in their implementation, and in light of the difficulties illustrated in this chapter, one can infer that current relational technology may not be the ideal way to manage CASE tool data. The AFIT SAtool scenario establishes that CASE tools could definitely benefit from the use of a DBMS because of the reduced data redundancy, crash recovery, concurrency control, and ease with which ad hoc queries can be made. Obviously, it's not yet clear as to whether the nested-relational DBMS will actually provide a tractable solution.

The chapter points out three commercialized databases explicitly designed for use by CASE tools. It looks at several attempts to provide CASE tool integration across the entire software life cycle; included were of AFIT's ongoing efforts in the area.

III. METHODOLOGY

Introduction

The IDEF₀ abstract data model and its relational and nested-relational DBMS implementations are presented in this chapter. After introducing certain notational adaptations to Chen's entity-relationship analysis methodology, an E-R analysis is used to develop an abstract data model of IDEF₀. This model helps mitigate some of the ambiguities inherent in IDEF₀. The model is divided into two parts representing the analysis data (the essential data model) and the graphical data (the drawing data model). This dual modeling approach allows for the extraction of analysis data without having to deal explicitly with the IDEF₀ graphical language. Relations corresponding to the E-R diagrams are then developed, stepwise refined, and mapped into a relational database design. An example database instance is developed. The relational design is implemented within Ingres Corporation's relational DBMS (Ingres) and loaded with the example data. SQL queries are developed to extract drawing data and essential data from the database. The relational design is transformed into a nested-relational design. The relational database instance is transformed into a nested-relational instance. SQL/NF queries are developed to extract drawing data and essential data from the nested-relational database.

Extended E-R Notation

As mentioned earlier, Codd's relational data model is the de facto database standard (7). However, several other models have been proposed, including Chen's entity-relationship (E-R) model (6). The E-R model includes the concepts of entities (represented by rectangles), attributes (represented by ellipses), relationships (represented by diamonds), and the links between them (represented by lines). One of the advantages of the entity-relationship model is that it allows for easy¹ mapping into a relational design. An E-R diagram based upon an example in Date

¹Many E-R design tools actually do this mapping.

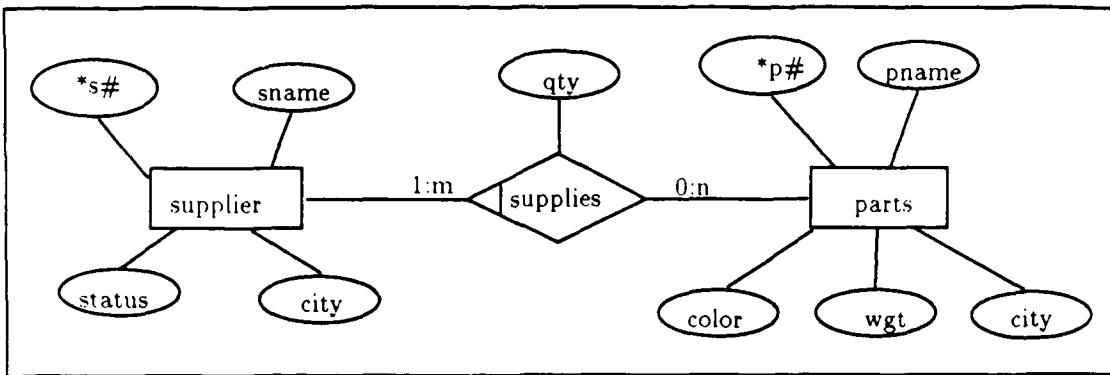


Figure 12. Modified Entity-Relationship Notation

is shown in Figure 12 (10). This drawing illustrates certain extensions to Chen's E-R notation which make the E-R diagrams more understandable by humans. In particular, a line is added on the side of the relationship construct to clarify how it relates to the corresponding entities. For example, Figure 12 is read "supplier supplies parts." Additionally, the cardinality is now explicit. for example, Figure 12 is read, "supplier supplies zero to many parts," and "parts are supplied by one to many suppliers." Finally, an asterisk is associated with the attribute which serves as the key, e.g., *s#* is the key attribute for entity, supplier.

IDEF₀ Abstract Data Model

In order to facilitate development of a DBMS implementation of the IDEF₀ language, an abstract data model of the language is constructed. This abstract data model consists of two parts, the essential data model, and the drawing data model; the concept is illustrated in Figure 13. This dual modeling approach allows for the extraction of analysis data without having to deal explicitly with the IDEF₀ language, i.e., without having to "walk through" the various drawings.

The IDEF₀ essential data model captures those portions of the language which represent the underlying semantics of a particular analysis (an IDEF₀ analysis could actually be represented by

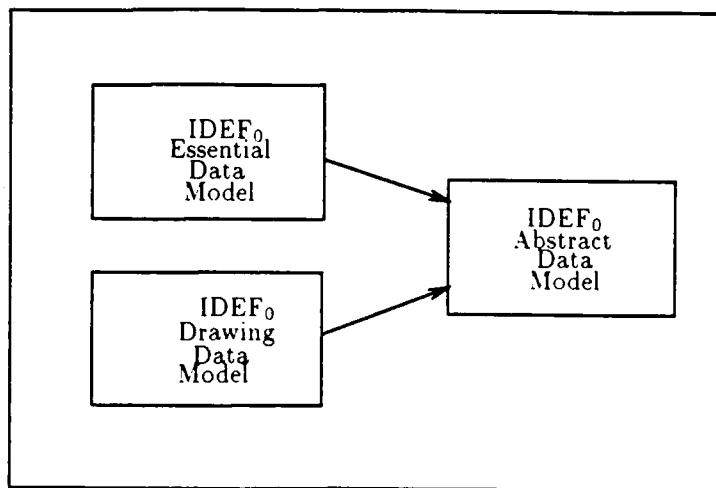


Figure 13. IDEF₀ Abstract Data Model: Essential Data and Drawing Data

infinitely many drawings by just moving one box a little on the diagram). Perhaps an analogy will better explain the concept. Say a good friend is going on a trip, and you wish to bid him goodbye. There are any number of different ways this could be done. A card that says "good riddance," a kiss, a handshake, a bon voyage party, etc. Now clearly, each of these actions is syntactically different, yet they all convey to the human the same semantics, i.e., "goodbye." In a similar sense, an IDEF₀ analysis may be syntactically expressed any number of ways, yet still convey the same semantical information to a human interpreter. It is this underlying "essential" data which is captured by the essential data model. This includes, for example, activities and their children, as well as data elements and their children. It does not include, for example, the location of the boxes or arrows which graphically represent the activities and data elements.

The drawing data model, on the other hand, encapsulates the graphical constructs which represent the particular IDEF₀ analysis. It contains such information as the location of the boxes which graphically represent activities, the line segments which graphically represent data elements, various other graphics artifacts, such as the location of "squiggles," the location of footnote markers.

some of the graphics "short-hand" such as double headed arrows, etc. This drawing data is used to actually draw an IDEF₀ diagram.

The entity-relationship method is used to represent both the essential data model and the drawing data model since it retains many of the semantics (for a human interpreter) of the actual data being modeled.

Essential Data Model. As described earlier, the IDEF₀ essential data model captures those portions of the language which represent the underlying semantics of a particular analysis. This includes, for example, activities and their children, as well as data elements and their children. It does not include, for example, the location of the boxes or arrows which graphically represent the activities and data elements.

In order to allow for an understandable, yet complete representation, the E-R analysis of the essential data model is done in two parts that complement one another. The first part shows the activity model, with the details about data elements left out. The second part shows the data element model while leaving out the details about the activities.

Figure 14 illustrates the essential model associated with IDEF₀ activities and Figure 15 illustrates the essential model associated with IDEF₀ data elements. Each of the entities and relationships for both E-R diagrams is explained in Table 1. Most of the attributes include a reference to show why the given attribute was necessary.

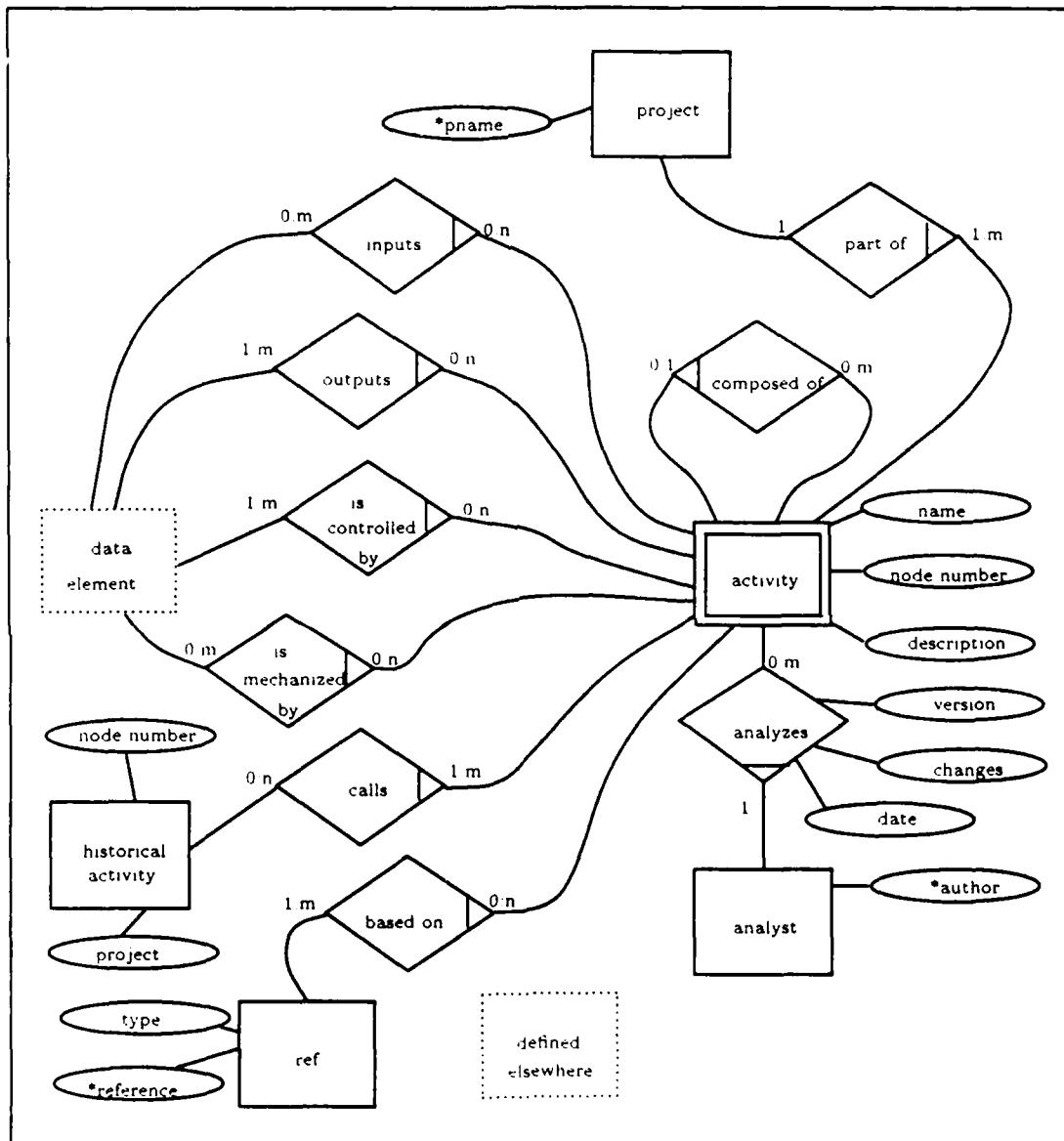


Figure 14. IDEF₀ ACTIVITY Essential Data Model

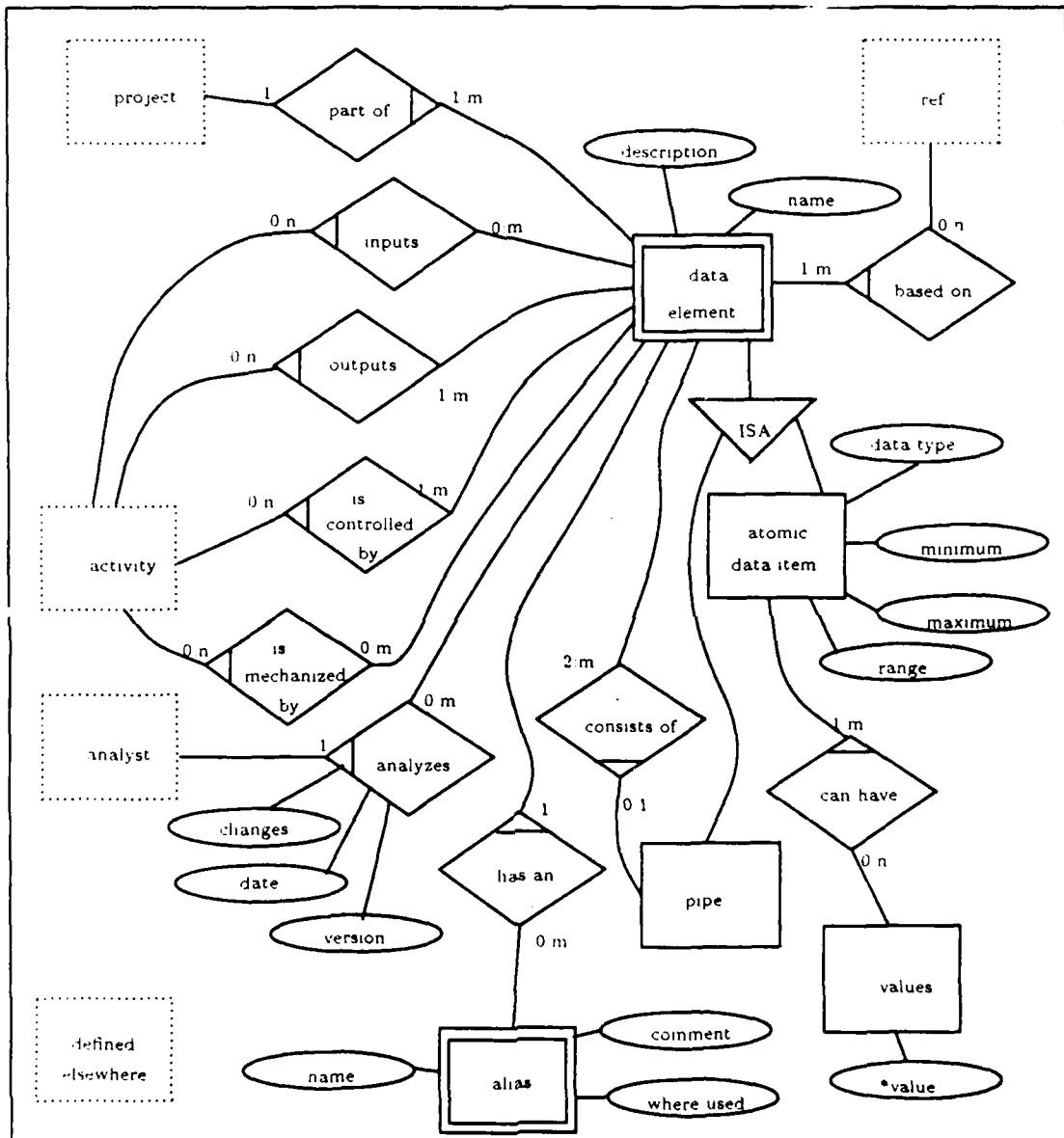


Figure 15. IDEF₀ DATA ELEMENT Essential Data Model

Table 1. Description of Components in the Essential Data Model

E-R construct	description
activity	This weak entity, which is existence dependent upon project , represents the IDEF ₀ activities. Attribute <i>node number</i> is the discriminant, and <i>name</i> captures the name of the given activity (24:13-14). Attribute, <i>description</i> , allows the analyst to describe the activity (16:12).
composed of	This relationship shows that a given parent activity is composed of zero to many (0:m) child activities. It also shows that each activity has one parent activity. The 0:1 notation accounts for the fact that the A-0 activity does not have a parent activity (16:12).
analyst	This entity is used to capture information about the analyst who performed the analysis. The reason for making analyst an entity, rather than an attribute of activity , is so that it might be tied into a personnel database. The entity, analyst , currently has the single attribute, <i>author</i> , which identifies the person who performed the analysis (16:12).
analyzes	This relationship expresses the fact that a given analyst analyzes zero to many activities (or data elements). Note that the current model only allows an activity (data element) to be analyzed by one analyst. Attribute, <i>version</i> , is used to record version information; <i>date</i> indicates when the analysis was performed; <i>changes</i> captures change information about a given activity (data element) (16:12).
project	This entity identifies the project to which each activity (data element) is assigned. Key attribute, <i>pname</i> , indicates the name of the project (16:12).
part of	This relationship indicates that an activity (data element) is part of exactly one project, whereas a project contains one to many activities (data elements).
ref	This entity captures any references associated with an activity (data element). Key attribute, <i>reference</i> , identifies which reference is involved, and attribute, <i>type</i> , identifies the type of reference (16:12). Basically, this entity allows a library of various documents such as DoD standards, user requirements, contractual clauses, etc., to be tied to the given activity (data element).
based on	This relationship indicates that a given activity (data element) is based on one to many references, and that a given reference is the basis for zero to many activities (data elements).
historical activity	This entity is primarily used as a convenience so that the database does not have to be loaded with analyses which were previously accomplished. Attribute, <i>project</i> , indicates which project contains the historical activity, and attribute, <i>node number</i> , identifies the specific activity within the project.

Table 1 (continued): Description of Components in the Essential Data Model

E-R construct	description
calls	This relationship indicates the fact that an activity can call zero to many previously completed (historical) activities, and that a given historical activity is called by one to many activities (30:33).
inputs	This relationship indicates that an activity can input zero to many data elements. Ross's SA (and the IDEF ₀ subset) only require activities to have control data elements and output data elements (30:20). Note that the entity, data element , is expanded in the next section.
outputs	This relationship shows that an activity must have at least one but can have many output data elements (30:22).
is controlled by	This relationship shows that an activity can have one to many control data elements (30:22).
is mechanized by	This relationship indicates that an activity can have zero to many mechanism data elements. Ross's SA (and the IDEF ₀ subset) only require activities to have control data elements and output data elements (30:20).
data element	This weak entity, which is existence dependent upon project , represents the IDEF ₀ data elements. Attribute, <i>name</i> , which is the name of the data element, is the discriminant (24:14). Attribute, <i>description</i> , allows the analyst to describe the data element (16:12).
pipe	This entity is a specialized data element, as illustrated via the ISA construct on the E-R diagram. It has no additional attributes, but merely indicates that the data element is actually a pipe containing at least two other data elements (30:20).
consists of	This relationship shows that a pipe consists of at least two data elements, and that a data element can be contained within at most one pipe.
atomic data item	This entity is also a specialized data element for capturing data that have atomic values, i.e., are not pipes. Attribute, <i>data type</i> , indicates the type of data (in the Pascal or Ada sense); <i>minimum</i> is the minimum data value, if applicable, <i>maximum</i> is the maximum data value, if applicable, and <i>range</i> is the data value range, if applicable (16:14). In the case that none of the attributes are applicable, entity values , as described below, probably applies.
values	This entity is used to accommodate atomic data items which have enumerated values, e.g., color can have values red, blue, and green. The entity has a single (key) attribute, <i>value</i> (16:14).
can have	This relationship ties the atomic data item entity to its corresponding values entity.
alias	This weak entity, which is existence dependent upon data element , captures any aliases that the given data element might have. Attribute, <i>name</i> , is the name of the alias and the discriminant of this weak entity set, attribute, <i>comment</i> , is used by the analyst to clarify why the alias was needed, and attribute, <i>where used</i> , indicates where the alias is used (16:14).
has an	This relationship shows that a data element can have zero to many aliases, and that a given alias corresponds to exactly one data element.

Drawing Data Model. As discussed above, the drawing data model represents the actual graphical constructs, e.g., boxes, line segments, etc., used to represent the particular IDEF₀ analysis.

As with the essential data model, the E-R analysis of the drawing data model is done in two parts that complement one another. The first part of the analysis shows the activities and other graphical constructs, e.g., squiggles, etc., with the details about data elements left out. The second part only shows the data element model.

Figure 16 illustrates the drawing model associated with IDEF₀ activities and Figure 17 illustrates the drawing model associated with IDEF₀ data elements. Each of the entities and relationships for both E-R diagrams is explained in Table 2. As appropriate, a reference is given citing why the entity, relationship, or attribute is needed.

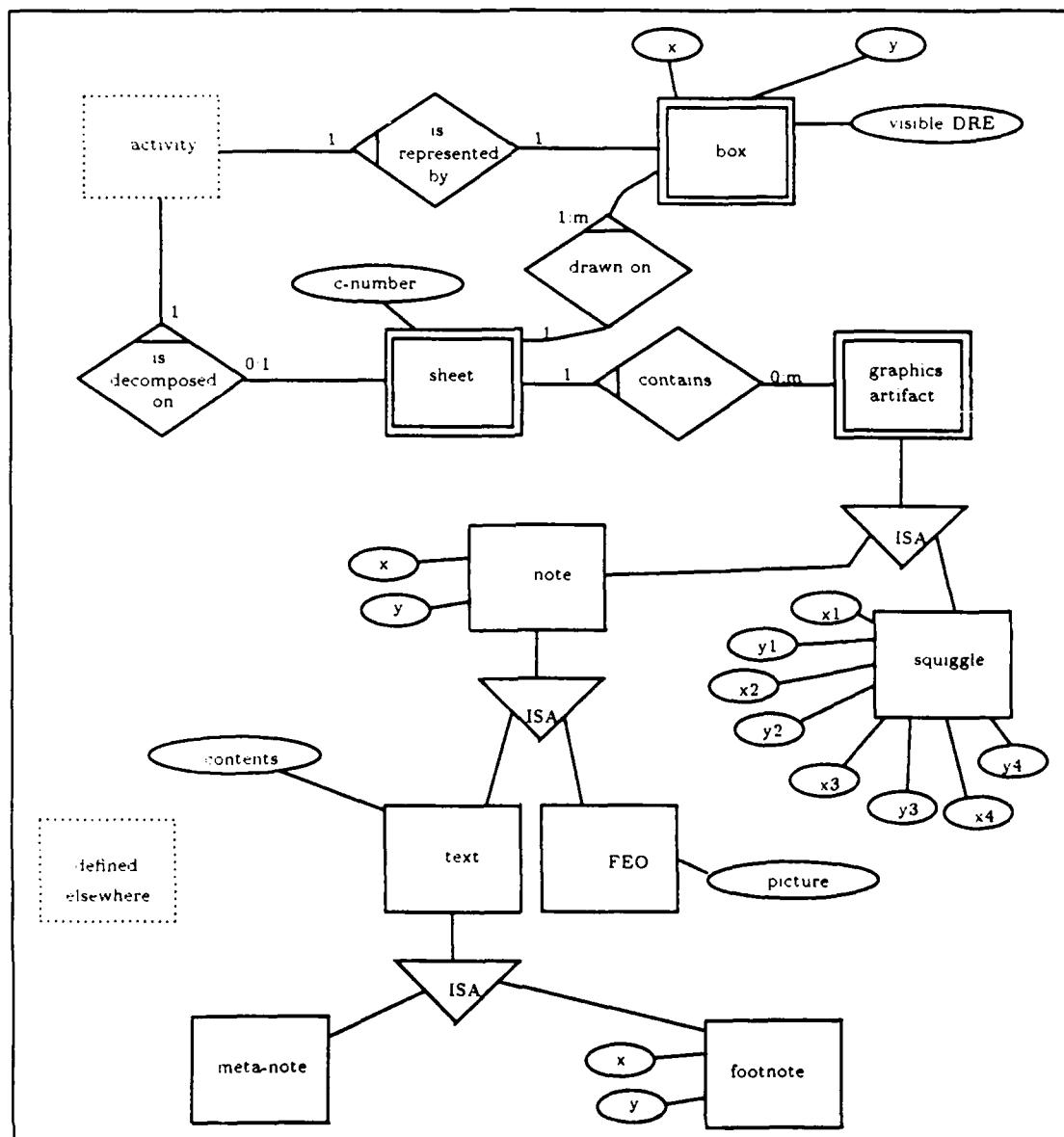


Figure 16. IDEF₀ ACTIVITY Drawing Data Model

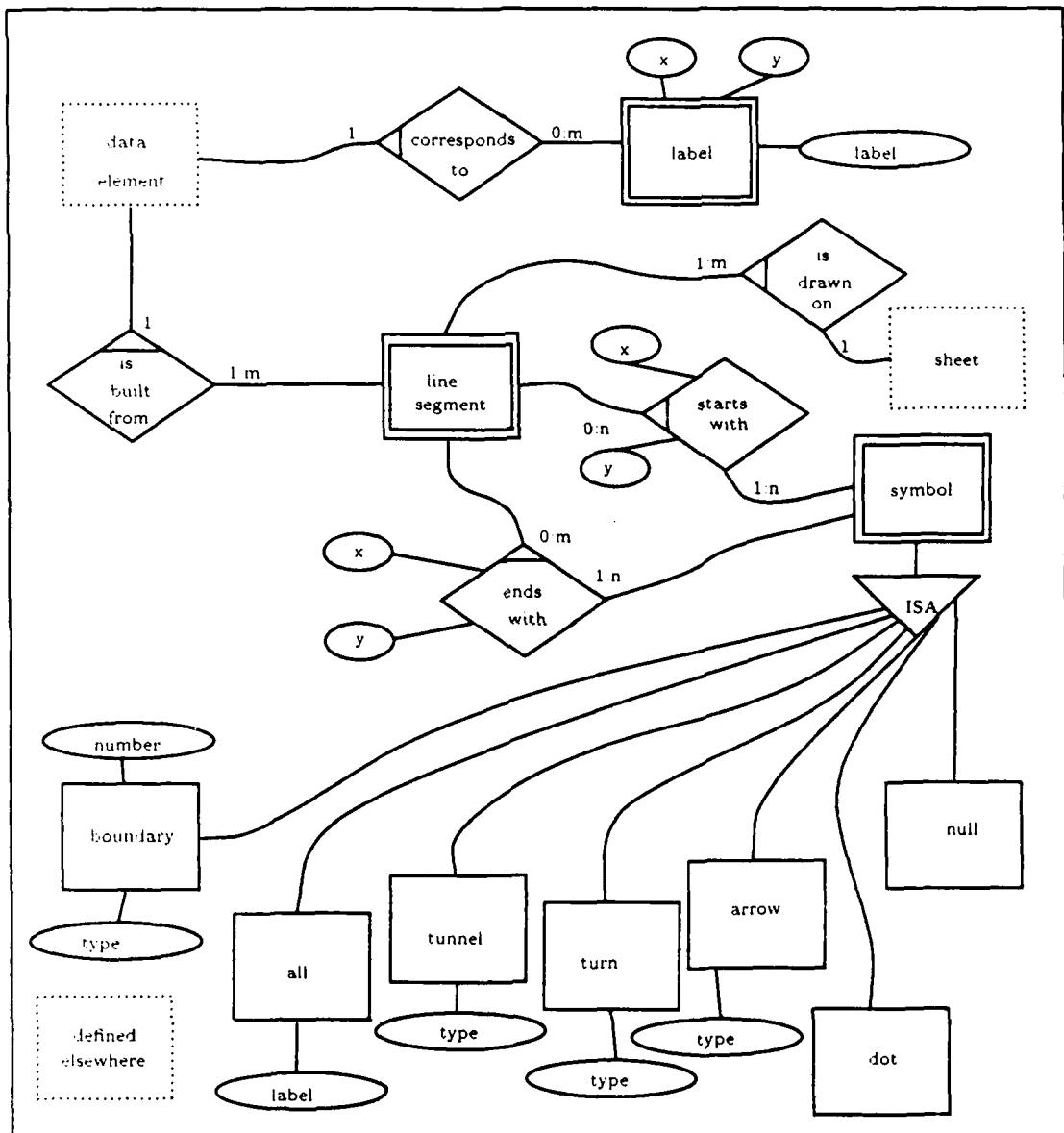


Figure 17. IDEF0 DATA ELEMENT Drawing Data Model

Table 2. Description of Components in the Drawing Data Model

E-R construct	description
box	This weak entity, which is existence dependent upon activity , captures the graphical construct which represents an activity on the IDEF ₀ diagram. Attributes, <i>x</i> , and <i>y</i> , indicate the location of the upper left hand corner of the box (all boxes are the same size). The attribute, <i>visible DRE</i> , corresponds to Ross's detail reference expression. In Ross's words "The omission of a detail reference expression indicates that the box is not further detailed in this model" (30:33).
is represented by	This relationship simply indicates the one to one correspondence between an activity and its graphical representation, box .
activity	This entity is described in the section dealing with the essential data model.
sheet	This weak entity, which is existence dependent upon activity , captures the fact that an activity is decomposed. It has the single attribute, <i>c-number</i> (24:17). Note that <i>c-number</i> is used as the DRE symbol on the parent diagram.
is decomposed on	This relationship ties an activity to the sheet upon which it is decomposed. if such a decomposition exists.
drawn on	This relationship ties a box to the sheet upon which it is drawn.
graphics artifact	This weak entity, which is existence dependent upon sheet is a generalized entity which includes note and squiggle (30:20).
contains	This relationship indicates that a given sheet can contain zero to many graphics artifacts.
note	This entity is used to capture the location of note markers, and it is the generalized entity for both text notes (footnote and meta-note), and FEO (30:20). There are two attributes, <i>x</i> , and <i>y</i> , which indicate the location of the note marker on the diagram.
squiggle	This entity simply contains the four ordered pairs which denote the location of a squiggle on the sheet (30:20).
text	This entity, which is a member of entity, note , as seen from the ISA construct, captures the text for meta-notes and footnotes. Attribute, <i>contents</i> , holds the text of the note and the ISA construct indicates the type of note. i.e., footnote or meta-note (30:20).
FEO	This entity, which is a member of entity, note , as seen from the ISA construct, captures the drawings associated with the for exposition only (FEO) (30:22).
footnote	This entity is a specialized type of text note as seen from the ISA construct. Attributes <i>x</i> , and <i>y</i> , are the location on the drawing where the footnote is placed.
meta-note	This entity is a specialized type of text note as seen from the ISA construct on the E-R diagram.
label	This weak entity, which is existence dependent upon data element , captures the label associated with a data element, as well as the location of the label on the diagram. Discriminant attribute, <i>label</i> , is the label itself, and attributes, <i>x</i> , and <i>y</i> , are the location of the first character of the label.

Table 2 (continued): Description of Components in the Drawing Data Model

E-R construct	description
corresponds to	This relationship connects a data element to its label. A data element can have zero to many labels, but a given label can only refer to one data element.
data element	This entity is described in the section dealing with the essential data model.
line segment	This weak entity, which is existence dependent upon data element , captures all the line segments from which the graphical representation of a data element is built.
is built from	This relationship simply indicates that a data element is graphically represented by at least one but perhaps many line segments.
is drawn on	This relationship indicates the sheet on which a particular line segment is drawn. It also indicates that a sheet can have one to many line segments drawn on it.
symbol	This weak entity, which is existence dependent upon line segment , is a generalized entity used to capture the type of symbol with which a line segment either starts or ends.
starts with	This relationship connects a line segment to the symbol with which the line segment starts. The two attributes, <i>x</i> , and <i>y</i> , are the location of the starting symbol. Note that a line segment can start with more than one symbol, e.g., an arrow and a dot.
ends with	This relationship connects a line segment to the symbol with which the line segment ends. The two attributes, <i>x</i> , and <i>y</i> , are the location of the ending symbol. Note that a line segment can end with more than one symbol, e.g., icon code (boundary) and arrow.
boundary	This entity indicates that the starting or ending symbol on the line segment corresponds to a boundary. Attribute, <i>type</i> , indicates the type of boundary (Input, Control, Output, and Mechanism), and attribute, <i>number</i> , is the number of the boundary (24:22).
all	This entity captures the to-all and from-all construct; the single attribute, <i>label</i> , captures the to-all/from-all label (30:31-32).
tunnel	This entity denotes that the line segment corresponds to a tunnel arrow; the attribute, <i>type</i> , indicates if this is an external arrow that did not appear on the parent diagram (hidden source), or if it is an arrow that touches an activity but does not appear on that activity's decomposition (hidden destination) (24:24).
turn	This entity is used if the line segment starts or ends with a turn. Attribute, <i>type</i> , determines the type of turn (right-up, left-up, right-down, left-down, up-right, up-left, down-right, and down-left).
arrow	This entity is used if the line segment starts or ends with an arrow. Attribute <i>type</i> determines the type of arrow (right, left, up, and down).
dot	This entity is used in the case of two-way arrows (30:20).
null	This entity is used if the line segment does not have a starting or ending symbol, e.g., the segment simply connects to another segment and therefore does not have a starting symbol.

IDEF₀ Relational Database

This section considers some of the design trade-offs associated with the relational implementation of the IDEF₀ database; the mapping of the E-R model into relations; an example database for the relational design; the Ingres implementation of this design; and SQL queries which extract data from the Ingres implementation.

Design Trade Offs. In coming up with the relational design, several trade offs are considered. With one exception, the relational design retains the explicit division between the essential data and drawing data. The exception is the relation, **activity**, which includes both essential data, and drawing data. The justification is simple. Fully separating the activity from its box representation results in either; a significant amount of replicated data, e.g., name (essential), name (drawing), node (essential), node (drawing), etc.; or requires all queries associated with retrieving box data to involve joins, which are costly. In short, the drawing data abstraction, **box**, and its essential data abstraction, **activity**, are both contained in the same relation.

In the interest of efficient queries, seemingly redundant "id" attributes are added to certain relations. For example, the attribute, *project_id*, in relation, **project**, is redundant since the attribute *name* is a superkey. However, a query involving a 4-byte integer *project_id* is intuitively more efficient (time wise) than a query involving a 12 byte character string.

All the relations corresponding to weak entities have a globally unique "id" attribute which eliminates the need for extraction of superkeys from the relation on which the given weak entity is existence dependent. This is done in the interest of efficiency (relative to queries). An example is the attribute, *node_id*, in the **activity** relation. Obviously the "real" superkey for **activity** is the attribute pair, *node*, *project_name*. Once again, a 4-byte integer comparison is more efficient.

Several of the drawing data relations have the *sheet_id* attribute included in them, whereas the E-R diagrams show no such direct relationship. This is done to eliminate the need for multiple joins. A good example is the **arrow** relation. Obviously, a join between **arrow**, **symbol**, **segment**,

and **sheet** is needed to determine which arrows go on which sheets. By simply adding the *sheet_id* into the **arrow** relation, the joins are eliminated. Clearly this generates additional problems relative to redundancy and consistency. However, it is felt that the *sheet_id* attribute is not like to change frequently, whereas queries to draw the diagrams will occur frequently. In short, the decision, while increasing redundancy and creating an increased potential for inconsistent data, provides a more efficient implementation (time wise).

Finally, some of the entities and relationships were collapsed into a single relation; some were renamed to make their roles more clear. This section includes tables which show the connection between the requirement (as manifested in the E-R analysis) and the implementation (as manifested in the relational design).

Relational Design. In developing the relational design, the approach taken was to initially do a straight one-to-one translation from the E-R diagrams, and then through stepwise refinement, reduce the tables as alluded to above. In fact, it took 21 iterations to get the relations into the format shown in Table 3. The **bold faced** header indicates the name of the relation and a brief description of the relation. This header is followed by a three column layout which shows the name of each attribute, its type (e.g., *integer4*), and a brief description of the attribute.

Table 3. Relational Design

act2act	cross reference parent activity to its child activities	
parent_node	i4	identifies the parent activity
child_node	i4	identifies the child activity
act2data	cross reference activity to its data elements	
node_id	i4	identifies the activity
data_id	i4	identifies the data element
icom_type	c1	I=input, C=control, O=output, M=mechanism
act2hist	cross reference activity to historical activity being called	
node_id	i4	identifies the calling activity
hist_id	i4	identifies called historical project/activity
act2ref	cross reference activity to its references	
node_id	i4	identifies the activity
ref_id	i4	identifies the reference
activity	IDEF ₀ activity or its box representation	
node_id	i4	key attribute identifies activity
node	c20	node number for activity
name	c25	name of activity
project_id	i4	identifies the project
author_id	i2	identifies the analyst
version	c10	activity version
date	c8	date this version was created
x	i2	x-wise location of box representation
y	i2	y-wise location of box representation
visible_DRE	i1	-1 = activity is decomposed, 0 = not yet
sheet_id	i4	sheet on which this activity appears
act_changes	changes made to the activity	
node_id	i4	identifies the activity
changes	c60	description of the change
act_descr	descriptions can be multiple lines	
node_id	i4	identifies the activity
line_no	i2	keep description lines in proper order
desc_line	c60	one line of the description

Table 3 (continued): Relational Database Design

alias	data element can have multiple aliases	
data_id	i4	identifies the data element
name	c25	name of alias
where_used	c25	where alias is used
comment	c25	why alias is needed
analyst	person doing the analysis	
author_id	i2	key attribute identifies analyst
author	c20	name of analyst
arrow	type of arrow	
symbol_id	i4	key attribute identifies arrow
arrow_type	i1	0 = up, 1 = down, 2 = left, 3 = right
boundary	ICOM codes for boundary arrows	
symbol_id	i4	key attribute identifies boundary symbol
icom_code	c2	I, C, O, M (input, control, output, mechanism)
data2data	cross reference pipe data element to its children	
parent_data	i4	identifies the parent data element
child_data	i4	identifies the child data element
data2label	cross reference data element to its labels	
data_id	i4	identifies the data element
label_id	i4	identifies the label
data2ref	cross reference data element to its references	
data_id	i4	identifies the data element
ref_id	i4	identifies the reference
data2value	cross reference data element to its data type values	
data_id	i4	identifies the data element
value_id	i4	identifies the data type value
data_changes	changes to the data elements	
data_id	i4	identifies the data element
changes	c60	description of the change
data_descr	descriptions can be multiple lines	
data_id	i4	identifies the data element
line_no	i2	keep description lines in proper order
desc_line	c60	one line of the description
data_elem	IDEF ₀ data element	
data_id	i4	key attribute identifies the data element
name	c25	name of data element
project_id	i4	identifies the project
author_id	i2	identifies the analyst
version	c10	data element version
date	c8	date this version was created

Table 3 (continued): Relational Database Design

data_range	range of this atomic data element	
data_id	i4	key attribute identifies the data element
data_range	c60	range of legal values
data_type	data type of this atomic data element	
data_id	i4	key attribute identifies the data element
type	c25	data type
data_value	legal values for enumerated atomic data elements	
value_id	i4	key attribute identifies the value
value	c15	the actual value
dot	type of dot (Ross's two-way arrow notation)	
symbol_id	i4	key attribute identifies dot
dot_type	i1	0 = above-right, 1 = below-right, 3 = below-left
footnote	location of actual footnote text	
graf_id	i4	key attribute identifies the footnote
x	i2	x-wise location of actual text
y	i2	y-wise location of actual text
feo	for exposition only (picture)	
graf_id	i4	key attribute identifies the FEO
picture	c60	perhaps the name of a graphics file?
graphics	graphics artifacts	
graf_id	i4	key attribute identifies graphic artifact
sheet_id	i4	identifies sheet on which graphic is drawn
hist_call	historical activity call	
hist_id	i4	key attribute
hist_proj	c12	name of project containing the historical activity
hist_node	c20	node number within the project, e.g., A312
label	label associated with a data element	
label_id	i4	key attribute identifies the label
name	c10	name of the label
x	i2	x-wise location of the label
y	i2	y-wise location of the label
sheet_id	i4	identifies sheet where label is drawn
min_max	minimum and maximum values for atomic data elements	
data_id	i4	key attribute identifies data element
minimum	c15	minimum value
maximum	c15	maximum value

Table 3 (continued): Relational Database Design

note	some kind of note	
graf.id	i4	key attribute identifies the note
label	c1	the label associated with the marker
x	i2	x-wise location of the marker
y	i2	y-wise location of the marker
note_text	contents of a note can be multiple lines	
graf.id	i4	key attribute identifies note contents
line_no	i2	keep contents lines in proper order
text_line	c60	one line of text for the note
project	project (model) name	
project_id	i4	key attribute identifies project (model)
name	c12	project name
reference	references can be multiple lines	
ref_id	i4	key attribute identifies the reference
line_no	i2	keep reference lines in proper order
ref_line	c60	a line of this reference
ref_type	type of reference	
ref_id	i4	key attribute identifies the reference
ref_type	c25	type of reference
segment	line segments graphically representing a data element	
seg_id	i4	key attribute identifies the line segment
data_id	i4	data element for this segment
sheet_id	i4	sheet where the segment is drawn
xs	i2	x-wise point where line segment starts
ys	i2	y-wise point where line segment starts
xe	i2	x-wise point where line segment ends
ye	i2	y-wise point where line segment ends
sheet	sheet containing activity	
sheet_id	i4	key attribute identifies the sheet
c_number	i4	Ross's c-number
squiggle	Ross's famous squiggle	
graf.id	i4	key attribute identifies the squiggle
x1	i2	location of the four points
y1	i2	which connect in the order
x2	i2	1 → 2 → 3 → 4
y2	i2	to make the squiggle
x3	i2	
y3	i2	
x4	i2	
y4	i2	

Table 3 (continued): Relational Database Design

symbol	segments start and end with many kinds of symbols	
symbol_id	i4	key attribute identifies the symbol
seg_id	i4	identifies the segment associated with this symbol
sheet_id	i4	identifies sheet on which the symbol is drawn
x	i2	x-wise location of symbol
y	i2	y-wise location of symbol
to_from_all	label for to-all and from-all arrows	
symbol_id	i4	key attribute identifies the symbol
tfa_label	c1	label in the to-all from-all circle
tunnel	Ross's famous disappearing arrows	
symbol_id	i4	key attribute identifies the symbol
tunnel_type	i1	-1 = hidden source, 0 = hidden destination
turn	type of turn, combos of up, down, right, and left	
symbol_id	i4	key attribute identifies the symbol
turn_type	i1	0=ru, 1=lu, 2=rd, 3=ld, 4=ur, 5=ul, 6=dr, 7=dl

Essential Data Requirements to Design Connection. Table 4 shows the connection between the essential data requirements (as manifested in the E-R analysis) and the implementation (as manifested in the relational design). The notation **entity.attribute**, and **relation.attribute** is used to denote a particular attribute for a given entity or relation.

Drawing Data Requirements to Design Connection. Table 5 shows the connection between the drawing data requirements (as manifested in the E-R analysis) and the implementation (as manifested in the relational design). As before, the notation **entity.attribute**, and **relation.attribute** is used to denote a particular attribute for a given entity or relation.

Example Relational Database Instance. An example database corresponding to the diagrams shown in Figure 7 and Figure 8 was constructed by hand. Appendix F has a listing of the example database.

Relational Implementation. The relational design shown in Table 3 is implemented within Ingres Corporation's relational DBMS, Ingres. The script file used to create the relations is shown

Table 4. Mapping of E-R Essential Data to Relational Design

E-R construct	Relational Design Construct
activity	activity
activity.description	act_descr
composed of	act2act
analyst	analyst
analyzes	activity.author_id data_elem.author_id
analyst.version	activity.version data_elem.version
analyst.date	activity.date data_elem.date
analyzes.changes	act_changes data_changes
project	project
part of	activity_project_id data_elem.project_id
ref	reference ref_type
based on	act2ref data2ref
historical activity	hist_call
calls	act2hist
inputs outputs is controlled by is mechanized by	act2data
data element	data_elem
data element.description	data_descr
pipe	data2data
consists of	
atomic data item.data type	data_type
atomic data item.minimum	min_max
atomic data item.maximum	
atomic data item.range	data_range
values	data_value
can have	data2value
alias	alias
has an	alias.data_id

Table 5. Mapping of E-R Drawing Data to Relational Design

E-R construct	Relational Design Construct
box is represented by	activity.x activity.x activity.y activity.visible_DRE
sheet	sheet
is decomposed on	To find the sheet on which an activity is decomposed, one must look at activity.sheet_id of any child of the activity (via act2act)
drawn on	activity.sheet_id
graphics artifact	graphics
contains	graphics.sheet_id
note	note
squiggle	squiggle
text	note.text
FEO	FEO
footnote	footnote
meta-note	A note tuple which does not have a corresponding footnote tuple is a meta-note.
label	label
corresponds to	data2label
line segment is built from is drawn on	segment
symbol starts with ends with	symbol
boundary	boundary
all	to_from_all
tunnel	tunnel
turn	turn
arrow	arrow
dot	dot
null	A line segment end for which there is not a symbol simply does not have an entry in any of the available symbol relations, e.g., arrow, etc.

in Appendix G, as are the script files used to input the bulk data files, show the contents of all the tables, and delete all data from the database. The next section shows some queries used to extract the drawing data from this example database.

SQL Queries. Although this thesis effort does not involve building a tool to actually draw an IDEF₁ diagram, such a tool might require the user to supply the name of the project. Accordingly, the queries shown below have "DM Example" as the project name.

The first query extracts the data required to begin drawing the A-0 diagram illustrated in Figure 7. The table immediately following the query contains the tuples that are extracted as a result of the query. Note that Ingres truncates the column title to be commensurate with the data size, e.g., **visible_DRE** is of type integer 1, so the column title gets truncated to **visibl**.

```
select a.x,a.y,a.name,a.date,an.author,a.version,a.visible_DRE,s.c_number
from activity a, analyst an, sheet s
where a.node = "A0" and
a.author_id = an.author_id and
s.sheet_id = a.sheet_id and
a.project_id in (
  select p.project_id
  from project p
  where p.name = "DM Example");

+-----+-----+-----+-----+-----+-----+
| x    | y    | name          | date      | author        | version    | visible|c_number |
+-----+-----+-----+-----+-----+-----+
| 1    | 1    | manage database | 102/14/89 | Gerald R. Morris | 1.0        | -1      | 1       |
+-----+-----+-----+-----+-----+-----+
```

At this point, a drawing tool can draw the blank sheet, and fill in NODE (A-0), NUMBER (always a 1 for A-0), PROJECT (DM Example), TITLE (same as PROJECT for A-0 sheet), DATE, AUTHOR, and REV (version) on the sheet. The tool can draw the box at location (x,y)², fill in the name of the activity, and enter the node number in the activity box³. Finally, if **visible_DRE** is true (-1), the tool can put the c-number to the lower right of the activity box to denote the sheet

² The (x,y) values are only symbolic in this example, normally they represent a location on the screen
³ In the specific case of the A0 node, the node number is a 1, not the expected 0, shame on you Douglas Ross!

AUTHOR: Gerald R. Morris	DATE 14Feb89	READER									
PROJECT DM Example	REV 1.0	DATE									
											
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 2px;">NODE</td> <td style="padding: 2px;">TITLE DM Example</td> <td style="padding: 2px;">NUMBER 1</td> </tr> <tr> <td style="padding: 2px;">A-0</td> <td></td> <td></td> </tr> </table>						NODE	TITLE DM Example	NUMBER 1	A-0		
NODE	TITLE DM Example	NUMBER 1									
A-0											

Figure 18. A-0 Diagram (partial drawing 1)

on which it is decomposed (Ross's detailed reference expression). The partial drawing that results is shown in Figure 18.

This next query extracts all the line segments which represent the data arrows on the A-0 sheet. As before, the table immediately following the query contains the tuples that are extracted as a result of the query.

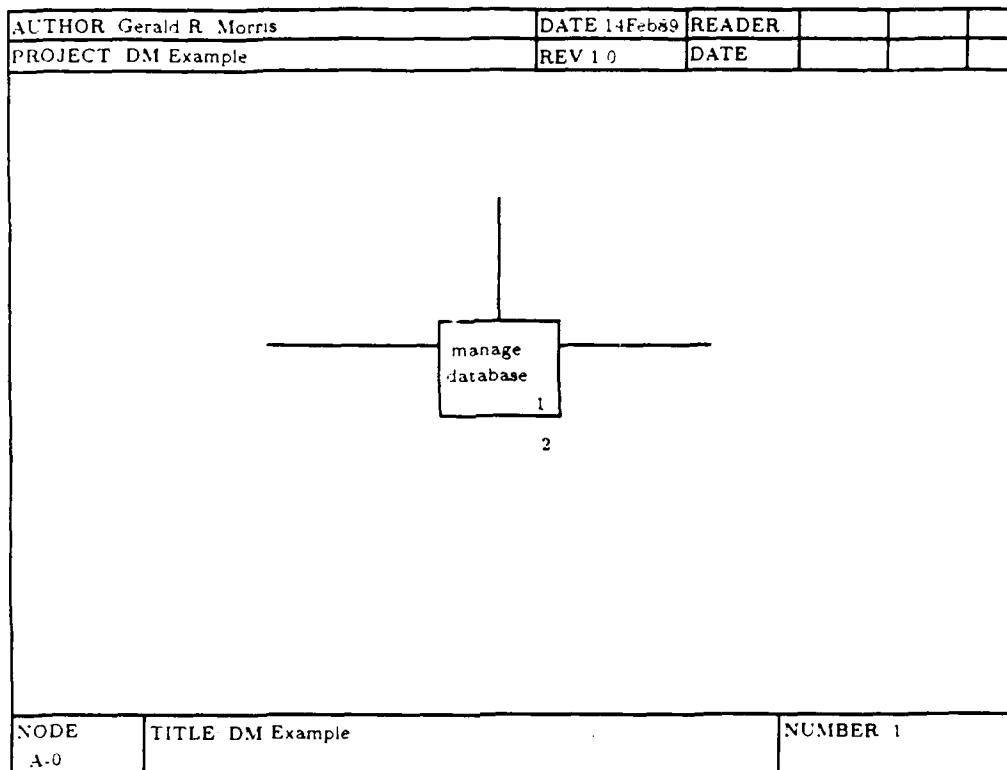


Figure 19. A-0 Diagram (partial drawing 2)

```

select se.xs,se.ys,se.xe,se.ye
from segment se
where se.sheet_id in (
  select a.sheet_id
  from project p,activity a
  where p.project_id = a.project_id and
  a.node = "A0" and
  p.name = "DM Example");

|xs      |ys      |xe      |ye      | | | |
|---|---|---|---|---|---|---|
| 3|      |3|      |4|      |4|
| 6|      |6|      |7|      |7|
| 9|      |9|      |10|     |10|
|-----|

```

Notice that for each line segment, there are two (x,y) pairs. These points represent the ends of the line segments which are drawn by the drawing tool. The updated partial drawing that results from adding the line segments is shown in Figure 19.

The queries required to complete the A-0 drawing are shown in Appendix G, as are the queries needed to draw the A0 diagram. In addition, Appendix G contains example SQL queries to extract essential activity data and essential data element data from the Ingres implementation. These queries provide the data necessary to create the data dictionary examples shown in Appendix C.

IDEF₀ Nested Relational Database

This section considers some of the design trade-offs associated with the nested-relational version of the IDEF₀ database; the mapping of the relational design into a nested-relational design; an example database for the nested-relational design; a "paper" implementation of this design; and SQL/NF queries which extract data from the nested-relational implementation.

Design Trade-Offs. Because the nested-relational model is an extension of Codd's relational model, the relational design is used as the starting point. In this nested-relational design, the drawing data and essential data are completely separated.

As with the relational design, certain "id" attributes are retained in the interest of efficient queries, e.g., a 4-byte integer comparison versus a 12-byte character string comparison.

There is some redundancy in this database, e.g., some of the data in the **activities** relation-valued attribute are also found in the **sheets** relation-valued attribute. The decision to do this is based on the cost of joins versus the cost of redundancy. Joins between nested relation-valued attributes can involve unnesting the involved relation-valued attributes, doing the join, and then nesting again. This is an intuitively expensive operation. It is felt that the cost of redundancy more than offsets the cost of doing a join on the nested attributes.

Several other design trade-offs are considered during the development of this nested-relational version. In particular; whether to nest data elements inside of activities, activities inside of data elements, or neither; whether to nest data elements inside data elements (recursively); and whether to recursively nest activities inside activities.

From a purely graphical viewpoint, the seemingly "natural" approach is to nest data elements inside of activities since a given activity's decomposition "contains" data elements. However, this quickly leads to trouble since a given data element is an input, output, control, or mechanism, perhaps a combination of several depending upon how many activities it touches. The implication of nesting data elements inside activities is that each activity which uses the data element has to nest the identical data in its attributes. This immediately leads to a potentially high degree of duplication. Thus there is again the dilemma of redundancy due to increased nesting versus increased requirements for joins due to a more flattened design. In a pathological case, a given data element might touch dozens of activities. Of course the biggest problem is that the essential data model does not keep track of tunnel arrows (a strictly graphical construct designed to minimize unnecessary clutter on a diagram). This means a given data element could "appear" suddenly as an input (for example) to an activity without having traversed the parent activity. In short, an update to a single data element requires an enumeration of all activities. Another problem that results from nesting data elements inside activities is that of deletion. If an activity is deleted, are all of its data elements also deleted? The issues described above relate to the partial dependency problem found in the flat relational model. A full discussion is beyond the scope of this thesis but Ozsoyoglu and Yuan address this problem within the context of nested-normal form (26). At any rate, the decision is to not nest data elements inside activities, to simply put a list of data element names and icom types inside each activity.

The next decision is whether or not to nest activities inside one another and whether or not to nest data elements inside one another. A fully nested approach in either case exhibits anomalous behavior relative to "when to stop" whenever queries are done because the algebra (query language) does not support recursion and transitive closure. Perhaps this is best illustrated by way of a simple example involving activities (a similar example can easily be constructed for data elements). Suppose that activities are recursively nested inside one another via relation-valued attribute, **child**. Further suppose a query is issued requesting all the offspring of the A0 node (in

flattened form) and that there are only three levels of nesting. A nested-relational algebra query looks something like

$$\pi(\text{node.name}(\sigma_\theta(\mu(\text{child}(\mu(\text{child}(\text{activity})))))))$$

where π is the database projection operation, σ is the selection operator, θ is a predicate identifying the appropriate project, and μ is the unnest operator (32) (8). The problem of course, in the general case, is that the number of levels of nesting associated with a particular node is not known apriori; thus, it is not known how to construct the query. On the other hand, a similar query for a design in which the activities are not nested inside one another could be expressed as

$$\pi(\text{node.name}(\sigma_\theta(\text{activity})))$$

where θ is a conjunctive predicate identifying the project and selecting all nodes starting with the sequence "A0".⁴ Thus, the decision is to keep activities at the same nesting level; the children are identified via a list of node names within each activity. For similar reasons, all data elements are kept at the same nesting level; children are identified via a list of data element names.

Nested-Relational Design. The IDEF₀ nested-relational schema consists of a single table, project. Unfortunately, the size of the project schema precludes the use of diagrams like those used by Roth (33:100), Colby (8:5), and others. Thus, in order to make the discussion of the scheme comprehensible, a hierarchical approach is necessary. Table 6 is a thumb-nail sketch of the hierarchical structure of the nested-relational design. To allow for an easier to comprehend representation a vertical presentation method is used.

⁴Recall the syntax of IDEF₀ requires offspring nodes to start with the parent node number, e.g., A221 is the first child of A22 and the second grandchild of A2. Thus we can take advantage of the node number relationships to determine hierarchical relationships.

Table 6. Nested-Relational Design

```

(project)
|project_name |
|-----|
(activities)
|node_id |node |name           |author           |version |date      |changes |c_number |parent  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
(act_descr)
|line_no |descr_line           |
|-----|-----|
(references)
|ref_type           |
|-----|
(ref_lines)
|line_no |ref_line           |
|-----|-----|
(hist_calls)
|hist_proj |hist_node |
|-----|-----|
(data_elems)
|data_name |lcom_type |
|-----|-----|
(children)
|node_name           |
|-----|
(data_elements)
|data_id |name           |author           |version |date      |changes |parent  |
|-----|-----|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no |descr_line           |
|-----|-----|
(references)
|ref_type           |
|-----|
(ref_lines)
|line_no |ref_line           |
|-----|-----|
(aliases)
|name    |where_used |comment   |
|-----|-----|-----|
(min_max)
|data_type |minimum    |maximum    |
|-----|-----|-----|
(range)
|data_type |range      |
|-----|-----|
(values)
|data_type |value      |
|-----|-----|
(activitees)
|node_name |lcom_type |
|-----|-----|
(children)
|data_name           |
|-----|

```

```

(sheets)
|c_number|node_name      |author      |version |date      |
|-----|-----|-----|-----|-----|
(boxes)
|node_name      |x |y |visible_DRE |
|-----|-----|---|-----|
(segments)
|data_id |
|-----|
(location)
|x |y |x |y |
|---|---|---|---|
(symbols)
|x |y |type_symbol |symbol_type |
|---|---|-----|-----|
(squiggles)
|x1 |y1 |x2 |y2 |x3 |y3 |x4 |y4 |
|---|---|---|---|---|---|---|---|
(meta_notes)
|label |x |y |
|-----|---|---|
(note_text)
|line_no |text_line |
|-----|-----|
(foot_notes)
|label |xm |ym |xn |yn |
|-----|---|---|---|---|
(note_text)
|line_no |text_line |
|-----|-----|
(feos)
|label |x |y |picture |
|-----|---|---|-----|
(labels)
|data_id |name      |x |y |
|-----|-----|---|---|

```

From the design sketch it is clear that, at a nesting level of 0, the **project** nested-relation scheme consists of four attributes as shown below. Relation-valued attributes (RVA) are indicated in **bold** type. atomic-valued attributes (AVA) are indicated in *italicized* type.

1. *project_name* - the name of the project
2. **activities** - all the activities associated with this project; this RVA captures essential activity data only
3. **data_elements** - all the data elements associated with this project; this RVA captures essential data element data only
4. **sheets** - the drawing data for the project; this RVA captures both the activity drawing data and the data element drawing data

The discussion that follows considers each of the three RVA, and their associated AVA/RVA.

At a nesting level of 1, the **activities** RVA scheme consists of fourteen attributes as listed below. As before, RVA are indicated in **bold** type, AVA are indicated in *italicized* type.

1. *node_id*⁵ - identifies the activity
2. *node* - node number, e.g., A32
3. *name* - name of the activity
4. *author* - name of the analyst
5. *version* - revision number
6. *date* - date of this revision
7. *changes* - changes for this revision
8. *c_number* - sheet on which this activity is drawn
9. *parent* - name of the parent activity
10. **act_descr** - description of the activity
11. **references** - why the activity is needed
12. **hist_calls** - calls to activities from other previously completed projects
13. **data_elems** - data elements associated with this activity
14. **children** - child nodes of this activity

The discussion that follows describes the RVA associated with the **activities** RVA. At a nesting level of 2, the **act_descr** RVA has 2 atomic attributes; *line_no* (helps ensure the description lines are retrieved in the proper order); and *desc_line* (one line in the description).

⁵The integer valued *node_id* is used in lieu of *node* for reasons of efficiency, i.e., it's faster to compare a 4-byte integer than a string.

At a nesting level of 2, **references** consists of two attributes; *ref_type* (the type of reference, e.g., MILSTD), and the relation-valued attribute **ref_lines** (contains lines of text elucidating the reference).

RVA **ref_lines**, at a nesting level of 3, has the two attributes *line_no*; and *ref_line*. The former is a number to help keep the latter lines of text in the proper order.

At a nesting level of 2, the **hist_calls** RVA, which has the two attributes *hist_proj* and *hist_act*, captures the project name and node (activity) which is being called.

The **data_elems** RVA (also at a nesting level of 2) has two attributes, *data_name* and *icomp_type* which specify the data element and its role (input, control, output, mechanism) with respect to the particular activity.

Finally, at a nesting level of 2 is the **children** RVA with a single attribute *node_name*. This is a list of all children of the activity⁶.

The next RVA considered is again at a nesting level of 1. The **data_elements** RVA scheme consists of fourteen attributes as enumerated below.

1. *data_id*⁷ - identifies the data element
2. *name* - the name of the data element
3. *author* - name of the person who did the analysis
4. *version* - the current version number
5. *date* - date of the revision
6. *changes* - changes reflected in this version
7. *parent* - name of the parent data element

⁶Strictly speaking, this is not necessary since a child node can always be found by looking at the node number, nonetheless it is included in the interest of efficiency.

⁷The integer valued *data_id* is used in lieu of name for reasons of efficiency, i.e., it's faster to compare a 4-byte integer than a string.

8. **data_descr** - a description of this data element
9. **references** - why the data element is needed
10. **aliases** - aliases for this data element
11. **min_max** - minimum and maximum values for atomic data elements
12. **range** - range of values for atomic data elements
13. **values** - list of values for enumerated atomic data elements
14. **activitees** - activities touched by the data element
15. **children** - children of pipe data elements

The discussion that follows describes the RVA associated with the **data elements** RVA. At a nesting level of 2, the **references** RVA consists of two attributes, **ref_type**, and **ref_lines**. The former indicates the type of references and the latter contains all the lines which comprise the given reference.

At a nesting level of 3, the **ref_lines** RVA consists of two attributes, **ref_type** which is used to keep the reference lines in proper order, and **ref_lines** which are the reference lines themselves.

At a nesting level of 2, the **aliases** RVA has three attributes. These attributes indicate the name of the alias (*name*), where it is used (*where_used*) and a comment line describing why the alias was needed (*comment*).

At a nesting level of 2, the **min_max** RVA has three attributes; *data_type* (the data type of this atomic data element); *minimum* (its minimum value); and *maximum* (maximum value). Note that this particular RVA will only have one tuple. The decision to make it an RVA is based on the fact that there are null values in the case of non-atomic data elements.

At a nesting level of 2, the **range** RVA has two attributes. These attributes indicate the data type of the atomic data element (*data_type*), and its range of values (*range*). Note that this

particular RVA will also only have one tuple. The decision to make it an RVA is again based on the fact that there are null values in the case of non-atomic data elements.

At a nesting level of 2, the **values** RVA has two attributes, *data_type* and *value*. The former identifies the data type of the atomic data element, and the latter indicates all legal values (this applies to enumerated data types).

At a nesting level of 2, the **activitees** RVA has two attributes, *node_name* and *icom_type*. The former identifies the activity touched by the data element, and the latter identifies the role of the data element (input,control,output,mechanism).

At a nesting level of 2, the **children** RVA contains a list of all children of this data element (if it's a pipe), via the single attribute, *data_name*.

Back again at a nesting level of 1, the **sheets** RVA scheme consists of 13 attributes as enumerated below. As before, RVA are indicated in **bold** type, AVA are indicated in *italicized* type.

1. *c_number* - the sheet identifier number
2. *node* - box being decomposed on this sheet
3. *name* - the title of the sheet
4. *author* - the person who drew the picture
5. *version* - the revision number
6. *date* - the date of the drawing
7. **boxes** - all boxes appearing on the sheet
8. **segments** - the line segments on the sheet
9. **squiggles** - squiggles on the sheet
10. **meta_note** - meta notes on the sheet

11. **foot_notes** - foot notes on the sheet

12. **feos** - for expositions only

13. **labels** - labels for line segments

The discussion that follows describes the RVA associated with **sheets**: At a nesting level of 3, the **boxes** RVA has 5 attributes. Attribute *node* is the node number, **name** is the name of the box, *x* and *y* are the location of the upper left corner of the box, and *visible_id* the c-number on which the box is decomposed or -1 if it is not decomposed.

At a nesting level of 2, the **segments** RVA scheme consists of three attributes *data_id*, **location**, and **symbols**, which indicate the data element being represented by the segment, and the location of the segment, and the symbols that appear at the ends of the line segment. As always, RVA are indicated in **bold** type, AVA are indicated in *italicized* type.

Immediately below **segments**, at a nesting level of 3, the **location** RVA scheme consists of four attributes *xs*, *ys*, *xe*, and *ye*, which indicate the start and end points of the lines segment(s) for the given data element.

Also below **segments** at a nesting level of 3, the **symbols** RVA scheme consists of four attributes: *x* and *y* indicate the location of the symbol; *type_symbol* is the type of symbol, i.e., arrow, boundary, dot, to-all/from-all construct, tunnel, or turn; and *symbol_type*, the interpretation of which depends upon *type_symbol*, indicates the symbol type, i.e., type of arrow, the icon_code, the type of dot, the label associated with a to-all/from-all construct, the type of tunnel arrow, or the type of turn.

At a nesting level of 2, the **squiggles** RVA scheme consists of eight attributes *x1*, *y1*, *x2*, *y2*, *x3*, *y3*, *x4*, and *y4*. These attributes indicate the four points which are connected together (in the order 1 → 2 → 3 → 4) to make Ross's squiggle.

At a nesting level of 2, the **meta_notes** RVA scheme consists of four attributes. Attribute *label* is the marker label for the meta_note, *x* and *y* are the location of the meta_note marker, and RVA **note_text** contains the text associated with the meta_note.

Immediately below **meta_notes**, at a nesting level of 3, is the **note_text** RVA which has two attributes, *line_no*, and *text_line*. The former attribute is used to ensure that the text lines in the latter attribute are retrieved in the proper order.

At a nesting level of 2, the **foot_notes** RVA scheme consists of six attributes; *label* indicates the foot_note marker label; *xm* and *ym* are the location of the foot_note marker; *rn* and *yn* are the location of the footnote text; and RVA **note_text** contains the actual lines of text in the footnote.

Immediately below **foot_notes**, at a nesting level of 3, is the **note_text** RVA which has two attributes, *line_no*, and *text_line*. The former attribute is used to ensure that the text lines in the latter attribute are retrieved in the proper order.

At a nesting level of 2, the **feos** RVA scheme consists of four attributes; *label* indicates the FEO marker label; *x* and *y* are the location of the marker; and *picture* is the name of the text/picture file associated with the FEO⁸.

At a nesting level of 2, the **labels** RVA scheme consists of four attributes. Attribute *data_id* identifies the data element associated with the label, *name* is the label itself, and *x* and *y* are the location where the label is drawn.

Example Nested-Relational Database Instance. An example database, containing the same information as the relational database, was constructed. As before, the intent was to determine if the nested-relational design had any "holes." Appendix H contains a listing of the example nested-relational database.

⁸In a nested database which supports objects of type bitmap (say), one could actually store the picture in the database.

Nested-Relational Implementation. Unfortunately, the Exodus-based nested-relational DBMS created by Mankus does not support all the capabilities required by this real world database application. Accordingly, a "paper" implementation of the nested-relational design is used. Certain extensions to SQL which support a nested-relational database have already been formalized by Roth and others (33). The queries of the nested-relational IDEF₀ database are developed using Roth's SQL/NF syntax⁹. The SQL/NF script to create the nested-relational schema is shown in Appendix I, as are the scripts to input the bulk data files into the nested-relational implementation, show all data, and erase all data.

SQL/NF Queries. The following query extracts drawing data from the nested-relational IDEF₀ database. This data is required by the drawing tool to create the A-0 diagram shown in Figure 7. As with the relational queries, the data resulting from this query is shown, in a postulated format, immediately following the query. The items delimited by parenthesis are the relation-valued attribute names.

```

SELECT (SELECT ALL BUT segments.data_id,labels.data_id FROM sheets WHERE node = "A-0")
FROM project
WHERE project_name = "DM Example";

(sheets)
|c_number|node|name|author|version|date|
|-----|-----|-----|-----|-----|-----|
|1|A-0|DM Example|Gerald R. Morris|1.0|102/14/89|
|-----|-----|-----|-----|-----|-----|
(boxes)
|node|name|x|y|visible_DRE|
|-----|-----|---|---|-----|
|AO|manage database|1|1|2|
|-----|-----|---|---|-----|
(segments)
(location)
|x|y|z|x|y|z|
|---|---|---|---|---|---|
|3|3|4|4| | |
|---|---|---|---|---|---|

```

⁹Since the nested-relational DBMS of Mankus is not used, the translation of SQL/NF to the Colby algebra is not required.

```

(symbols)
|x |y |type_symbol |symbol_type |
|---|---|-----|-----|
|4 |4 |arrow |right_arrow |
|---|---|-----|-----|
|---|---|---|---|
|6 |6 |7 |7 |
|---|---|---|---|
|---|---|-----|-----|
|7 |7 |arrow |down_arrow |
|---|---|-----|-----|
|---|---|---|---|
|9 |9 |10 |10 |
|---|---|---|---|
|---|---|-----|-----|
|10 |10 |arrow |right_arrow |
|---|---|-----|-----|
(squiggles)
|x1 |y1 |x2 |y2 |x3 |y3 |x4 |y4 |
|---|---|---|---|---|---|---|---|
|12 |12 |13 |13 |14 |14 |15 |15 |
|---|---|---|---|---|---|---|---|
(meta_notes)
|label |x |y |
|-----|---|---|
|-----|- |---|
(note_text)
|line_no |text_line |
|-----|-----|
|-----|-----|
(foot_notes)
|label |xm |ym |xn |yn |
|-----|---|---|---|---|
|1 |11 |11 |90 |90 |
|-----|---|---|---|---|
(note_text)
|line_no |text_line |
|-----|-----|
|1 |an example decomposition |
|2 |not completed |
|-----|-----|
(feos)
|label |x |y |picture |
|-----|---|---|-----|
|-----|---|---|-----|
(labels)
|name |x |y |
|-----|---|---|
|userdata |2 |2 |
|rules |5 |5 |
|feedback |8 |8 |
|-----|---|---|

```

Appendix I includes additional SQL/NF scripts which extract the drawing data for the A0 diagram, as well as essential data for a typical activity, and a typical data element.

Summary

An IDEF₀ abstract data model is presented in this chapter. The justification for a dual modeling approach (essential data and drawing data) is developed. The relations corresponding to the E-R diagrams are developed, as is an example database. The relational design is implemented in Ingres Corporation's relational DBMS, Ingres. SQL queries to extract drawing data from the relational database are constructed, and the data is used to actually draw some IDEF₀ diagrams. Additional queries to extract essential data are constructed. The relational design is transformed into a nested-relational design. An example database is constructed. SQL/NF queries to extract data from the nested-relational database are constructed and used to draw IDEF₀ diagrams. Additional queries to extract essential data are constructed.

IV. FINDINGS

Introduction

This chapter summarizes the IDEF₀ implementation within a relational and nested-relational DBMS by comparing the two implementations in several areas including query complexity, size of the database, and speed of query execution.

Query Complexity

This section considers the complexity of queries for the relational versus nested-relational versions of the IDEF₀ database. In particular, it looks at the queries associated with IDEF₀ drawing data and essential data.

A Definition of Complexity. Obviously a database query language complexity measure could include such criteria as number of joins, number of projects, number of binary comparison operators, number of unions, number of nests, number of unnests, etc. Unfortunately, some of these criteria do not necessarily apply to SQL/NF, and others do not apply to SQL. It is kind of like comparing apples to oranges. Thus, in order to ensure some type of commonality, assume that complexity is just a simple count of select-from-where (SFW) clauses. This conservative approach actually favors SQL over SQL/NF as suggested by the following example:

Suppose that r_1 is a 1NF relation on scheme R_1 , and r_2 is a 1NF relation on scheme R_2 , where $R_1 = (a, b, c)$, and $R_2 = (b, d, e)$. Suppose it is possible to generate a nested-relation, r_3 , on scheme, R_3 , where $R_3 = (a, B, c)$ (B is a relation-valued attribute such that $B = (d, e)$). The query of interest is the entire database. The queries are:

```
select r1. a, r1. b, r1. c, r2. d, r2. e /* SQL */  
from r1, r2  
where r1. b = r2. b.  
  
select ALL from r3, /* SQL/NF */
```

Using the simple complexity measure suggested above, both queries appear to have the same complexity since they both have a single SFW clause. It is easy to show that the SQL query, which involves a natural join, is more complex, both from a user viewpoint, and an internal execution viewpoint. The user viewpoint is obvious; an intimate knowledge of the matching keys, as well as the syntax for creating a natural join are required. The internal viewpoint is based upon some knowledge of how the nested-relational implementation works. Basically, any efficient nested-relational implementation will keep all the tuples in a nested relation on contiguous blocks of the disk. The Exodus storage manager allows for this to occur. It defines a storage object as "an uninterrupted container of bytes which can range in size from a few bytes to hundreds of megabytes" (5.1). In short, the nested-relational version requires the storage manager to return one or more contiguous blocks from the disk. The relational version, on the other hand, will have to use some type of join strategy, e.g., block-oriented iteration, to generate the required tuples. Generally this results in multiple disk accesses from non-contiguous blocks, even if both relations have clustered indices on the join attribute.

Thus, by using the simple criteria above, the results are conservative. That is to say, the extent to which SQL/NF appears to be "better" than SQL is understated.

The complexity measure discussed above can be formalized as, $C_{sfu}(query) =$ 'the number of SFW clauses in $query$ '. Note that C_{sfu} is additive; if extraction of a data set involves n queries, then the complexity of the query for the entire data set is given by

$$C_{sfu}(total) = \sum_{i=1}^n C_{sfu}(query_i)$$

where $C_{sfu}(query_i)$ is the complexity of the i 'th query.

Data definition language (DDL) statements, and data manipulation language (DML) statements, which may not involve SFW clauses, need additional complexity definitions.

In the case of DDL statements, the complexity is given by $C_{ddl} = n_a + n_t$, where n_a is the number of atomic attributes, and n_t is the number of TABLE clauses.

For DML statements which perform a bulk load, the complexity is given by $C_{bulk} = n_a + n_{rva} + n_c$ where n_a is the number of atomic attributes, n_{rva} is the number of relational-valued attributes, and n_c is the number of COPY TABLE clauses.

For DML statements which delete data, the complexity is defined as $C_{del} = 1 + n_{sfu}$, since there is always at least one DELETE keyword, and n_{sfu} SFW clauses ($n_{sfu} = 0$ is allowed).

The SQL and SQL/NF scripts in this thesis do not include any INSERT or UPDATE DML statements. Neither do they involve NEST or UNNEST. Accordingly, complexity measures associated with these types of statements are not defined.

Comparison of SQL versus SQL/NF. Having formalized the meaning of "complexity," it is now possible to compare the relational SQL queries with the nested-relational SQL/NF queries. A direct comparison of the drawing data and essential data queries illustrate rather profoundly the simplicity of the SQL/NF queries¹ as shown in Table 7. From a more intuitive viewpoint relative to the nested version, all the data is nested exactly where it is needed, the requirement for joins is minimized/eliminated, as is the requirement for multiple queries from multiple relations.

Table 7. Comparison of Query Script Complexity

Query	C_{sfu}	
	SQL	SQL/NF
A-0 activity drawing data	16	1
A0 activity drawing data	49	1
A1 activity essential data	7	1
unumber data element essential data	14	1
<i>average</i>	21.5	1

¹Recall these numbers are conservative.

The create tables scripts for SQL and SQL/NF are on the same order of complexity since, at some point, all the atomic attributes must be defined. Even so, because of the many to many relationships in the IDEF₀ abstract data model, the relational version needs to have a number of relations just to resolve the many to many conflict. The load tables scripts are also on the same order of complexity since, once again, all atomic attributes must be referenced. On the other hand, the erase tables script for SQL/NF is clearly less complex since there is only one table to erase! The actual complexity measures are shown in Table 8.

Table 8. Comparison of DDL/DML Script Complexities

	C_{add}	C_{load}	C_{del}
SQL	161	137	40
SQL/NF	113	119	1

Size of Database

This section considers the relative sizes of the relational implementation and nested-relational implementation of the IDEF₀ database. Unfortunately, a direct comparison is not possible since the nested-relational DBMS build by Mankus does not have all the features necessary to implement a complex nested-relational application such as IDEF₀. Fortunately, the logical comparison is still possible and is presented below.

Relational Logical Size. In the relational instance the algorithm for determining logical size is to count the number of bytes per tuple in each relation, determine the number of tuples, and then multiply the two. The example shown in Table 9, taken from Roth, illustrates the idea (33:102).

Suppose attribute, *dno* requires 4 bytes; *dname* requires 15; and *loc* requires 10. Assume there are 10 tuples in the database. Since each tuple uses 29 bytes and there are 10 tuples, the logical size of the database is $29 \times 10 = 290$.

A similar analysis of the IDEF₀ relational database instance is shown in Table 11.

Table 9. Simple Relational Example
Dept

dno	dname	loc
10	Manufacturing	Austin
20	Personnel	Dallas
30	Retail	Austin
:	:	:

Nested-Relational Logical Size. In the nested-relational instance, the approach is to count the number of bytes used by the *atomic* attributes in each relational-valued attribute, count the total number of tuples in each RVA, and then multiply the two. The simple example shown in Table 10, taken from Roth, illustrates the idea (33:100).

Table 10. Simple Nested-Relational Example
Supply

supplier	Supplies
	part
42	7
	8
	9
	10
	18
	20
	21
45	8
	10
	32
	34
	38
56	3
	5
	10
	41

Suppose the *supplier* atomic attribute uses 5 bytes, and the *part* atomic attribute uses 3 bytes. Since there are 3 occurrences of a **Supply** tuple and 16 occurrences of a **Supplies** tuple, the logical size of the database is $5 \times 3 + 3 \times 16 = 63$. A similar analysis of the IDEF₀ nested-relational database is shown in Table 12.

The latter results bear some explanation since, in general, nested-relations are small then their relational counterparts. To make joins more efficient in the relational design, globally unique integer "id" attributes were used in lieu of the "real" superkeys. Since the nested-relational version avoids joins as much as possible, it would have been counterproductive to include such "id" attributes. Not including them naturally increases the size of the nested-relational model since all the attributes must be stored. This is perhaps best understood by way of some examples.

Suppose that the **activity** relation had stored the 12 character project name as the join attribute instead of the 4 byte integer, *project_id*, the resulting size of an **activity** tuple would be 90 bytes as opposed to 82 bytes. In the case of **data_elem**, the size of a tuple would have been 61 bytes as opposed to 53 bytes. The space savings associated with the 2 byte integer, *author_id*, as opposed to the 20 byte character string, *author*, is 18 bytes per tuple for both **activity** and **data_elem**. Perhaps the biggest savings is in the cross reference tables. For example, assume **activity** used its "real" key, *project_name*, which is a 12 byte character string, and *node*, which is a 20 byte character string, in lieu of the 4 byte integer key, *node_id*. Further suppose that **data_elem** used *project_name*, and *data_name*, which is a 25 byte character string, in lieu of the 4 byte integer, *data_id*. Consider the effect on the **act2data** cross reference relation. Since attribute *room_code* remains the same, but the other two attributes, *node_id*, and *data_id* are changed as just described, one tuple of **act2data** would require 70 bytes instead of 9 bytes. In short, the size savings associated with the relational version has to do with the design decision to use global integer "id" values in lieu of the "real" keys; it is not an intrinsic feature of the relational model. Thus, the nested-relational instance, which uses 6,579 bytes, takes up more room than the relational instance, which only uses 6,086 bytes.

Table 11. Logical Size of Relational Instance

RELATION	BYTES	TUPLES	STORAGE
act2act	8	2	16
act2data	9	12	108
act2hist	8	1	8
act2ref	8	4	32
activity	82	3	246
act_changes	64	0	0
act_descr	66	6	396
alias	79	0	0
analyst	22	1	22
arrow	5	12	60
boundary	6	3	18
data2data	8	6	48
data2label	8	14	112
data2ref	8	9	72
data2value	8	2	16
data_changes	64	0	0
data_descr	66	13	858
data_elem	53	11	583
data_range	64	2	128
data_type	29	3	87
data_value	19	2	38
dot	5	0	0
feo	64	0	0
footnote	8	1	8
graphics	8	2	16
hist_call	36	1	36
label	22	14	308
min_max	34	0	0
note	9	1	9
note_text	66	2	132
project	16	1	16
reference	66	19	1,254
ref_type	29	13	377
segment	20	21	420
sheet	8	2	16
squiggle	20	1	20
symbol	16	26	416
to_from_all	5	3	15
tunnel	55	1	55
turn	5	8	40
TOTAL STORAGE			6,086

Table 12. Logical Size of Nested-Relational Instance

RELATION/RVA	ATOMIC BYTES	TUPLES	STORAGE
PROJECT	12	1	12
activities	176	3	528
act_descs	62	6	372
refrences	25	4	100
ref_lines	62	6	372
hist_calls	32	1	32
data_elems	26	9	234
children	25	2	50
data_elements	152	11	1,672
data_descr	62	14	868
refrences	25	9	225
ref_lines	62	12	744
aliases	75	0	0
min_max	55	0	0
range	86	0	0
values	30	2	60
children	25	6	150
sheets	87	2	174
boxes	51	3	153
segments	4	14	56
location	8	21	168
symbols	8	26	208
squiggles	16	1	16
meta_notes	5	0	0
note_text	62	0	0
foot_notes	9	1	9
note_text	62	2	124
feos	65	0	0
labels	18	14	252
TOTAL STORAGE			6,579

Speed of Query Execution

This section considers the speed of query execution for the relational implementation and nested-relational implementation of the IDEF₀ database. Unfortunately, a directly measurable comparison is not possible since the nested-relational DBMS build by Mankus does not have all the features necessary to implement a complex nested-relational application such as IDEF₀. However, it is still possible to compare the implementations from two different perspectives. The first approach assumes that the applicable tuples are fetched from the disk as needed. Speed is determined based upon a conservative assumption as to the number of disk accesses needed. The second approach assumes that the entire data set associated with a given project is small enough to fit in main memory. Speed is determined via an order-of analysis of a typical program that talks to the database via embedded language queries. The initial load at run time, and final flush at termination time are ignored.

Disk Resident Project Data. In order to determine the relative speeds of the queries, certain conservative estimates are made relative to physical mapping of the data onto the disk.

Assume that the relational DBMS has memory resident clustered B+ tree indices on the join attributes. Further assume that the DBMS is "smart" enough to store relations that are likely to be joined in close proximity to one another on the disk. Based upon these highly optimistic assumptions, assume that it takes one disk access for every two relations involved in a join. Formally, the number of disk accesses, n_{sql} , for a relational (SQL) query is given by

$$n_{sql} = \lceil \frac{n_r}{2} \rceil$$

where n_r is the number of unique relations involved in the query being considered.

Assume that the nested-relational DBMS stores the data in contiguous locations on the disk, and that it takes two accesses to retrieve a given set of tuples for the queries considered. The

latter is actually a fairly good assumption. The example database instance shown above is around 8 kilo-bytes, which includes three activities, eight data elements, and two sheets. Considering that a 2k to 4k page size is typical, it seems reasonable to presume a query for a given sheet, activity, or data element could be done with just two accesses. As to the contiguous storage assumption, Exodus definitely allows this capability as discussed in the Exodus Storage Manager guide (5:1). In short, the assumptions seem to be reasonable. The last assumption is that a memory resident index identifies the appropriate disk locations. Formally, the number of disk accesses, n_{sqlnf} , for the given nested-relational (SQL/NF) queries is given by

$$n_{sqlnf} = 2.$$

Table 13 shows the results for the queries given in Chapter 3:

Table 13. Relative Query Speeds: Number of Disk Accesses

Query	n_{sql}	n_{sqlnf}
A-0 activity drawing data	17	2
A0 activity drawing data	36	2
A1 activity essential data	13	2
unumber data element essential data	25	2
<i>average</i>	22.75	2

Memory Resident Project Data. Obviously a tool which uses the IDEF₀ database needs to talk to the database. The mechanism with which a high-level language such as C or Ada interfaces with the resident DBMS is the embedded query language call. The next two sections consider such embedded query language calls for the Ingres relational IDEF₀ implementation and the nested-relational implementation. The assumption is that the entire dataset for a given project is small enough to fit into main memory. Note that certain liberties are taken relative to syntax. The important issue here is the concept (semantics), not the syntax!

Embedded SQL Example. Embedded SQL in Ingres involves the concept of a cursor (28:3-7). Basically, a "template" SQL statement is created, along with some data structures that are "visible" to SQL. Whenever the embedded SQL call is made, the next tuple is placed in the visible data structures. Consider the following C header file which defines the types needed to contain the screen data in the IDEF₀ database (an analogous set of Ada type definitions is given in Appendix J):

```

/* This header defines the data structures that are used to capture the
   drawing data from the IDEF0 database via embedded query language calls
   It is not known apriori how many tuples there are, so a linked list
   structure is used.

   The element names correspond identically to the attribute names used in
   the IDEF0 database. It is assumed the user of this header is familiar
   with the database schema... */

typedef struct box{
    char node[21],name[26];
    int x,y,visible_dre;
    struct box *next_box;      /* pointer to next box */
} *boxptr;                      /* type boxptr points to a box structure */

typedef struct loc{
    int xs,ys,xe,ye;
    struct loc *next_loc;
} *locptr;

typedef struct symbol{
    int x,y;
    char type_symbol[13],symbol_type[13];
    struct symbol *next_symbol;
} *symbolptr;

typedef struct seg{
    locptr location;
    symbolptr symbols;
    struct seg *next_seg;
} *segptr;

typedef struct squig{
    int x1,y1,x2,y2,x3,y3,x4,y4;
    struct squig *next_squig;
} *squigptr;

typedef struct note_txt{
    int line_no;
    char text_line[1..61];
    struct note_txt *next_note_txt;
} *note_txtptr;

```

```

typedef struct meta{
    char label[2];
    int x,y;
    note_ttxptr note_text;
    struct meta *next_meta;
} *metaptr;

typedef struct foot{
    char label[2];
    int xm,yx,xn,yn;
    note_ttxptr note_text;
    struct foot *next_foot;
} *footptr;

typedef struct feo{
    char label[2];
    int x,y;
    char picture[61];
    struct feo *next_feo;
} *feoptr;

typedef struct label{
    char name[11];
    int x,y;
    struct label *next_label;
} *labelptr;

typedef struct sheet{
    int c_number;
    char node[21],name[26],author[21],version[11],date[9];
    boxptr boxes;
    segptr segments;
    squigptr squiggles;
    metaptr meta_notes;
    footptr foot_notes;
    feoptr feos;
    labelptr labels;
} *sheetptr;

/* and some other stuff as well */

```

Given this header file, which defines the data structures needed to draw the screen, a procedure must now make the appropriate embedded SQL calls, load the resulting tuples in the screen drawing structure, and then call the screen drawing procedure. The following elided routine illustrates the concept:

```

/* Obligatory procedure header, includes, et al */

draw_the_a_minus_zero_screen(the_project_name,...) /* relational version */
/* required parameter declarations */
.

{
    /* perform necessary initialization to talk to Ingres SQL */
    ...

    /* declare C variables that are visible to SQL. Note that strings
       gotta be one more since C uses a null byte to teminate strings */
    EXEC SQL BEGIN DECLARE SECTION;
        char project_name;           /* name of project */
        /* declare the variables for the first query */
        int x,y,                   /* coordinates of box */
            visible_dre,           /* true if this box is decomposed */
            c_number;               /* sheet on which it is decomposed */
        char name[26],              /* activity name */
            date[9],                /* date of the revision */
            author[21],              /* name of analyst */
            version[11];             /* revision number */
        /* declare the variables for the remaining queries */
        ...
    EXEC SQL END DECLARE SECTION;

    /* Create some local linked list structures to hold all the stuff
       until you can allocate the screen drawing data structure.
       Recall, we don't know its size yet... */
    ...

    /* let SQL "see" the name of the project */
    strcpy(project_name, the_project_name);

    /* Open Database */
    ...

    /* Create query cursor, basically this is a prototype of the
       desired SQL query that is to be performed. This
       cursor is used to generate the first part of the A-0 sheet */
    EXEC SQL DECLARE a_minus_0_first_cursor CURSOR FOR
        select a.x,a.y,a.name,a.date,an.author,a.version,a.visible_DRE,s.c_number
        from activity a, analyst an, sheet s
        where a.node = "A0" and
              a.author_id = an.author_id and
              s.sheet_id = a.sheet_id and
              a.project_id in (
                  select p.project_id
                  from project p
                  where p.name = :project_name);

    /* Open cursor and prepare to perform the query */
    EXEC SQL OPEN a_minus_0_first_cursor;

    EXEC SQL WHENEVER NOT FOUND GOTO duni; /* Branch when done - Ugh! */

    /* Query loop */
    while(1) { /* until the stinking goto above is taken */
        /* Fetch next tuple via defined cursor and put the resulting
           tuple into the C variables defined above */

        EXEC SQL FETCH a_minus_0_first_cursor INTO
            :x, :y, :name, :date, :author, :version, :visible_dre, :c_number;

        /* allocate a new node to save this data */
        ...
    }
}

```

```

        /* fetch next tuple (none in this case) */
    }

duni: /* No more tuples; do whatever clean up is necessary */

        /* Close cursor */
        EXEC SQL CLOSE a_minus_0_first_cursor;

/* Define the next cursor
   and grab the data in a query loop
   and at each pass, allocate a new local node to keep the data,
   and load it into the new node */

        ...

/* Define the next cursor
   ...ad nauseum... */

        ...

/* Close database */

        ...

/* now that the data is available, create, allocate, and load a sheetptr
   type data structure to send to the drawing routine */
sheetptr the_sheet;

        ...

/* now that the data is loaded, call the draw screen procedure */
draw_screen(the_sheet);

} /* that's all folks */

```

In order to assess the worst case time complexity of this code, it is necessary to look at the definition of Big-O and order-of:

Definition of Big-O:

A function $f(n)$ is of order $O(g(n))$ if and only if there exist constants, $c > 0$ and $n_0 \geq 0$, such that

$$f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$

$f(n) = O(g(n))$ says that $g(n)$, multiplied by some constant, c , gives an upper bound on $f(n)$. (21)

The code shown above would actually involve ten different cursors and query loops, as seen in Appendix G. The first loop always has exactly one tuple, and is therefore of time order, $O(1)$. The second loop is of time order, $O(s)$, where s is the number of segments on the A-0 diagram. The third loop is of time order, $O(a)$, where a is the number of arrow symbols. The fourth loop is of time order, $O(t)$, where t is the number of turn symbols. The fifth loop is of time order, $O(u)$.

where u is the number of tunnel arrow symbols. The sixth loop is of time order, $O(l)$, where l is the number of labels. The seventh loop is of time order, $O(q)$, where q is the number of squiggles on the A-0 diagram. The eighth loop is of order, $O(f)$, where f is the number of footnotes. Although not shown in the example A-0 diagram in this thesis, a generalized A-0 drawing might include meta-notes and FEO constructs. These would require two additional loops, which would run on the order of, $O(m)$ (number of meta-notes), and $O(p)$ (number of FEO pictures associated with the diagram). In all the cases above, the constants would depend upon the machine used, system loading (in a multiuser environment), etc. Nonetheless the order-of analysis, in conjunction with the fact that there are 10 sequential loops, does give an indication that the embedded SQL queries could be potentially slow. This rings particularly true since for each of the 10 queries, there is a separate parsing, optimization, and extraction. In addition, the queries all involve a multi-way join. As discussed in the previous section, this is likely to be an expensive process.

Embedded SQL/NF Example. The prototype nested-relational DBMS built by Mankus does not support embedded SQL/NF. Accordingly, postulate the existence of such a capability. Assume it is along the same lines as the embedded SQL in Ingres. A call to embedded SQL/NF gets the next tuple, much like Ingres embedded SQL. However, since a "single tuple" contains relational-valued attributes, the embedded SQL/NF must automatically allocate space in a linked-list structure which receives the data. In short, assume that embedded SQL/NF requires a cursor, which is again a "template" SQL/NF statement, and some linked-list data structure which is "visible" to SQL/NF. Whenever the embedded SQL/NF call is made, the next tuple is placed in the linked list along with all tuples in its relational-valued attributes.

Assume the header file mentioned in the previous section is available. The following elided routine illustrates the embedded SQL/NF call:

```

/* Obligatory procedure header, includes, et al */

draw_the_a_minus_zero_screen(the_project_name,...)
/* nested-relational version */
/* required parameter declarations */
. . .

{
    /* perform necessary initialization to talk to SQL/NF */
    . . .

    /* declare C data structures that are visible to SQL/NF.  */
    EXEC SQLNF BEGIN DECLARE SECTION;
        char project_name;           /* name of project */
        sheet_ptr the_sheet;         /* the entire enchilada! */
    EXEC SQLNF END DECLARE SECTION;

    /* let SQLNF "see" the name of the project */
    strcpy(project_name, the_project_name);

    /* Open Database */
    . . .

    /* Create query cursor, basically this is a prototype of the
       desired SQLNF query that is to be performed.  This
       cursor is used to generate the entire A-0 sheet */
    EXEC SQLNF DECLARE a_minus_0_cursor CURSOR FOR
        SELECT (sheets ALL BUT segments.data_id,labels.data_id WHERE node = "A-0")
        FROM project
        WHERE project_name = :project_name;

    /* Open cursor and prepare to perform the query */
    EXEC SQLNF OPEN a_minus_0_cursor;

    /* Fetch next tuple via defined cursor and put the resulting
       tuple into the the_sheet defined above */
    EXEC SQLNF FETCH a_minus_0_cursor INTO :the_sheet;

    /* Close cursor */
    EXEC SQLNF CLOSE a_minus_0_cursor;

    /* Close database */
    . . .

    /* Now that the data is loaded, call the draw screen procedure */
    draw_screen(the_sheet);
}

/* that's all folks */

```

In the case of SQL/NF, there is a single cursor and query (recall there is a single tuple per sheet). Accordingly, the procedure is of time order, $O(1)$. Now obviously the constant for a nested query will be different than the constant for a non-nested query. Even so, there is a single parsing, optimization, and extraction. Accordingly, it seems reasonable to suggest that the SQL/NF query will run faster. Obviously, the actual speed depends a great deal upon the particular machine, system loading, etc.

Summary

This chapter summarizes the IDEF₀ implementation within a relational and nested-relational DBMS.

A conservative definition of query complexity is presented, and used to determine the relative complexities of SQL (relational) queries, and SQL/NF (nested-relational) queries. The average complexity of the relational (SQL) queries is an order of magnitude (21.5 times) more complex than the average complexity of the nested-relational (SQL/NF) queries.

The logical size of the relational and nested-relational instances are derived. The larger size of the nested-relational instance (6,579 bytes) as compared to the relational instance (6,086 bytes) is due to the use of integer "id" attributes in the relational design; it is not due to some intrinsic quality of relational versus nested-relational designs.

The speed of execution for queries is presented from two aspects; a disk based DBMS wherein the number of disk accesses determines the speed; and a memory resident DBMS wherein the speed of execution is determined via program run time.

Disk-based relational (SQL) queries require an average of 22.75 disk accesses. Disk-based nested-relational (SQL/NF) queries require an average of 2 disk accesses.

An order-of analysis is used to determine the relative speeds of queries for the memory resident DBMS. A program containing embedded SQL (relational) queries to extract the A-0 drawing data runs in linear time based upon the number of graphical constructs in the diagram. A program containing embedded SQL/NF (nested-relational) queries to extract the A-0 drawing data runs in constant time. Obviously the constants associated with the order-of analysis are dependent upon the particular machine, operating system, number of users, etc.

V. CONCLUSIONS AND RECOMMENDATIONS

Introduction

This chapter summarizes and presents conclusions about the research; it also includes some recommendations as to further research in this area.

Summary

This research effort accomplished several objectives relative to the design of both a relational and nested-relational database to handle the IDEF₀ analysis language data. The primary accomplishments include the following:

1. Developed a partitioned abstract data model of IDEF₀ which included:
 - An essential data model.
 - A drawing data model.
2. Developed a relational DBMS to handle the IDEF₀ language data which included:
 - Mapping the abstract data model into a relational design.
 - Implementing the relational design in the Ingres DBMS.
 - Creating an example database instance.
 - Developing SQL queries which extract:
 - Drawing data.
 - Essential data.
3. Developed a nested-relational DBMS to handle the IDEF₀ language data which included:
 - Mapping the relational design into a nested-relational design.
 - Creating an example database instance.

- Developing SQL/NF queries which extract:
 - Drawing data.
 - Essential data.

4. Compared the relational and nested-relational versions in terms of:

- Complexity of queries.
- Size of the database.
- Speed of query execution.

Conclusions

The partitioned abstract data model, while more complex, allowed the drawing data to be separated from the essential data.

In the nested-relational design, the drawing data and essential data were completely separated. In this sense, the nested-relational version more closely modeled the abstraction generated via the E-R analysis.

The complexity results, which were based on metrics that favor SQL over SQL/NF, clearly showed the advantage of the nested-relational version over the relational version for the particular set of queries considered. On average, the relational queries (SQL) were an order of magnitude (21.5 times) more complex than the equivalent nested-relational (SQL/NF) queries.

The nested relational instance had a slightly larger logical size when compared to the relational instance. It was shown that this was a result of optimizing the relational design via use of global integer "id" attributes instead of the "real" keys. Even so, the apparent benefits of simpler queries and faster execution times would seem to offset the increased size. Obviously from a user perspective this is desirable (assuming the storage is available).

The query speeds of the disk-based DBMS model clearly showed the advantage of the nested-relational version over the relational version. On average, the relational (SQL) queries required 22.75 disk accesses whereas the nested-relational queries only required an average of 2 disk accesses. This is an order of magnitude speedup. In large part, this is due to the assumption of a contiguous storage model for the nested-relational DBMS. An object-oriented storage model, such as Exodus (5), does allow this capability.

The query speeds for the memory-based DBMS model also showed that the nested-relational version has an advantage over the relational version. The running time of the embedded SQL/NF program which extracts the A-0 drawing data is of order, $O(1)$; the running time of the embedded SQL program is of order, $O(\max(s, a, t, u, l, q, f, m, p))$, where s is the number of segments, a is the number of arrow symbols, t is the number of turn symbols, u is the number of tunnel arrow symbols, l is the number of labels, q is the number of squiggles, f is the number of footnotes, m is the number of meta-notes, and p is the number of for-exposition-only picture constructs. Obviously the constants associated with big-O depend upon the machine used, system loading (in a multiuser environment), etc. Nonetheless the order-of analysis, in conjunction with the fact that there are 10 sequential loops in the relational case, does give an indication that the embedded SQL queries could be potentially slow relative to the nested-relational case. This rings particularly true since for each of the 10 queries, there is a separate parsing, optimization, and extraction. In addition, the queries all involve a multi-way join.

The overall conclusion is that a nested-relational data model has an advantage over a relational model for this particular application (IDEF₀), and the particular queries considered (drawing data and essential data).

Recommendations

Unfortunately, this research effort generated more questions than answers. Obviously these questions can only be answered through additional research.

The most obvious area where additional work needs to be done is in an actual nested-relational implementation of the derived design. Given a robust nested-relational DBMS, it would be possible to implement both a relational version and a nested-relational version using the same DBMS. In the relational case one would simply create tables that only have atomic valued attributes. The advantage of using the same DBMS is that the confounding influence attributable to different machines, algorithms, data structures, etc., would be minimized. Once the two implementations are in place, one could then develop a set of database instances/queries from which statistically valid conclusions could be made relative to run times, query complexities, size of the database, etc.

The thesis investigation does not consider either the relational or nested-relational version in terms of input and update. This area needs to be addressed.

The assumptions as to number of disk accesses for the relational versus nested-relational versions needs to be empirically studied.

Chapter 2 discussed several commercially available DBMS that are designed explicitly for use within a CASE tool environment. An interesting research topic might be to see if one of these DBMS more appropriately meets the needs of the IDEF₀ abstract data model. In addition, chapter 2 looks at some efforts to integrate CASE tool data across the software development life cycle. In particular, the EDIF standardized file interchange solution, and the ATIS "generic DBMS" solution. An interesting research topic might be to map the IDEF₀ data requirements into one of these proposed formats in order to allow the data to be available during all phases of the software development process.

Another potential research area concerns an embedded query language interface to SAtool II. One suggestion is to build an Ingres-based embedded SQL version of SAtool II. The interface

should be as modular as possible such that the nested-relational SQL/NF version could be plugged in later. Considering that SAtool II is written in Ada, this modularization should be possible.

The complexity measures developed in this thesis effort did not include all of the SQL/NF syntax, and they ignored the complexity introduced by joins, unions, compound predicates, etc. An interesting research effort might be to develop some metrics by which the complexity of various queries can be compared. Obviously these parametrically based metrics would have to be statistically validated using a sample from some population of queries.

Appendix A. Some CASE Tools and Vendors

The following descriptions represent some of the more popular CASE tools currently on the market. It by no means is an exhaustive list. Nonetheless, it does give some idea as to the variety of different products and vendors.

Adagraph *Analytic Science, Arlington, VA* Combines early graphical design with an Ada style PDL. Users can define graphics idioms for tasks and subtasks, and then insert and reuse these idioms in their designs. The tool provides general idioms for recurring tasks such as buffering, monitoring, data movement or device driving.

Aris *Software Systems Design, Claremont, CA* Takes an analysis result from the Cadre Teamwork tool and automatically creates a first-cut at an Ada program structure. Working from dataflow diagrams, Aris also does a preliminary partitioning.

Autocode *Integrated Systems, Santa Clara, CA* Developed out of a need to simulate control system operation. Working somewhat like chips, Autocode software blocks contain prewritten code. Each block performs a different transformation on its inputs. The user basically wires the blocks together on the screen. As with most other code generators, some of the code must be hand written, e.g., the code for reading of inputs and posting of outputs to outside hardware.

Byron PDL *Intermetrics, Cambridge, MA* An Ada design language that lets the user add keywords and comments in the code thus making it possible to automatically extract DoD-STD-2167 documentation from the annotated code via the Byron Document Generator.

Designaid *Nastec, Southfield, MI* Based upon the Ward Mellor methodology. The user draws the diagrams, which the system then checks to ensure that the connections are legal, and that the data items have been defined correctly. All verified information is automatically stored in the system database.

EPOS-S *Software Products and Services, New York, NY* Combines graphics and PDL. It is a design-oriented formal language used for partitioning the software design and for more detailed design using a PDL. Users start with graphics at the higher levels, and then switch to PDL at the lower levels. There is a module to check for completeness, consistency and module interconnection. There are templates for DoD-STD-2167 documentation.

Excelerator *Index Technology, Cambridge, MA* Supports both Yourdon DeMarco and Gane Sarson systems analysis methods; supports Chen or Merise entity-relationship analysis methods. With this tool, users can develop data flow diagrams and structure charts. The tool can be used to develop data model diagrams, entity relationship diagrams. It allows the user to generate presentation graphics to present the overall system definition to users and managers. All information is kept in Excelerator's data dictionary, which is one of Excelerator's most powerful features. Every part of a graph can be described to the central project database at the time you create it. This "record-as-you-go" feature prevents loss or unnecessary duplication of data. Index has done quite well with its IBM-PC based version of Excelerator. It is probably the most well known and perhaps the most capable of the current generation of PC based CASE tools.

Popkin Windows System Architect *Chelsea Systems, New York, NY* Supports the Ward Mellor methodology. The user draws diagrams, the system checks them for compliance with a set of rules, and stores the results in a data base

Promod/RT *Promod, Lake Forest, CA* Based on the Hatley control-flow and state-transition diagrams. The user draws the required graphics with the tool's graphics editor, and then enters control specifications et al, by way of a text editor. The results are then saved in files. The tool can perform an automatic provisional design partitioning based upon these files created by the software analysis.

Ready Taskbuilder *Ready Systems, Palo Alto, CA* Set of tools based on Ward Mellor method. Allows users to develop data-flow diagram, get information on intertask synchronization and communication, estimate concurrent execution timing, define data items, data types and Ada-style software packages, lay out a graphical design, and then proceed to write program design language (PDL) descriptions.

Reverse Engineering *Meta Systems, Ann Arbor, MI* Examines code or data dictionaries and then automatically forms a logical view of the software. Users can look at the calling structures of the code, get data definitions and data structures, and locate dead data and code.

Software Engineering Workbench *Yourdon, New York, NY* Based on Cadware's Rule Tool. Yourdon has added its own icons and associated rules for the Ward Mellor real-time systems method, global checking, and its own data dictionary.

Statemate *iLogix, Burlington, MA* Interesting tool in that it can execute real-time software specifications. The simulation can be for the entire system or any syntactically complete portion of the system considered in isolation. The system can also generate DoD-STD-2167 documentation for use by companies developing software under DoD contracts.

Structured Architect-Real Time *Meta Systems, Ann Arbor, MI* Implements the Ward Mellor software-analysis methods. This tool generates the required graphics including data-flow, data-control, and state-transition diagrams, and state transition matrices. Users also create structured lists and data base entries. This material is used by SA-RT to automatically enter information into a data base.

SuperCASE *Advanced Technology International, New York, NY* Prompts the user to name the Ada-like subsystems and packages. That information is used to generate subsystem, package and task template specifications—all of which can be customized or used as is. Then the designer moves on to templates for package, subpackage and task bodies, writing and editing them in PDL on the screen-displayed templates.

Tags *Teledyne Brown Engineering, Fairfax, VA* A requirements language to express software specifications. It uses blocks and icons to create timing and flow diagrams. These specifications can be checked to uncover static errors and then executed to simulate the real-time operation of the software being modeled. The executable code generated from the description is in the Ada language, but is for use only in this kind of dynamic checking, not in the final system.

TekCASE Designer *Tektronix, Beaverton, OR* Tool set which implements the Ward Mellor or Hatley method for system software analysis. It merges real-time software specification diagrams into a single diagram and produces an editable provisional partitioning, for software design. A listing and evaluation routine reports on inconsistencies, problem areas and deviations from defined structured design methods. TekCASE does not do automatic code generation, but the output of the listing routine can be edited into code stubs for the language being used. Once the source code is completed, a TekCASE Designer routine can convert it into structure charts, compare the charts with the final design and report on any differences.

Toolkit *Cadre Group, New Haven, CT* Can be adapted to a several popular diagramming methods. The user selects icons with the mouse or keyboard and uses them to draw the diagrams. If a use-rule is violated, a circled X appears over the affected item and an error message is displayed. Another routine automatically enters information into a database from information drawn on the screen. The Verify routine detects any global violations of rules, particularly

those violations that couldn't be checked during the drawing process. There's also a tool (Cadware Rule Tool) that lets users modify existing analysis methods or set up their own methods for creating software diagrams

Appendix B. *IDEF₀ Language Features*

Ross defines 40 features of his Structured Analysis language, which "constitute the basic core of the language for communication" (30:19). Unfortunately, the *IDEF₀* user's manual does not include a single summary of the *IDEF₀* language as does Ross's paper. However, during his thesis effort, Johnson extracted the *IDEF₀* subset from the user's manual and cross referenced it to the appropriate SA feature. The following table is adapted from Johnson's work:

Table 14. *IDEF₀ Language Features*

SA Item No.	Name
1	box
2	arrow
3	input
3	output
4	control
5	mechanism
6	activity name
7	label
12	branch
13	join
14	bundle
15	spread
18	boundary arrow
20	detailed reference expression
22	2-way arrow
24	tunnel arrow
25	to/from all
27	footnote
28	metanote
29	squiggle
30	c-number
31	node number
32	model name
33	ICOM code
37	facing page text
38	for exposition only
39	glossary
40	node index

(19:A-3)

Appendix C. SAtool Products

The following pages present some typical examples of the current SAtool products, as illustrated in Figure 20. Recognize that the new Ada version, built by Smith, may have a slightly different format (35).

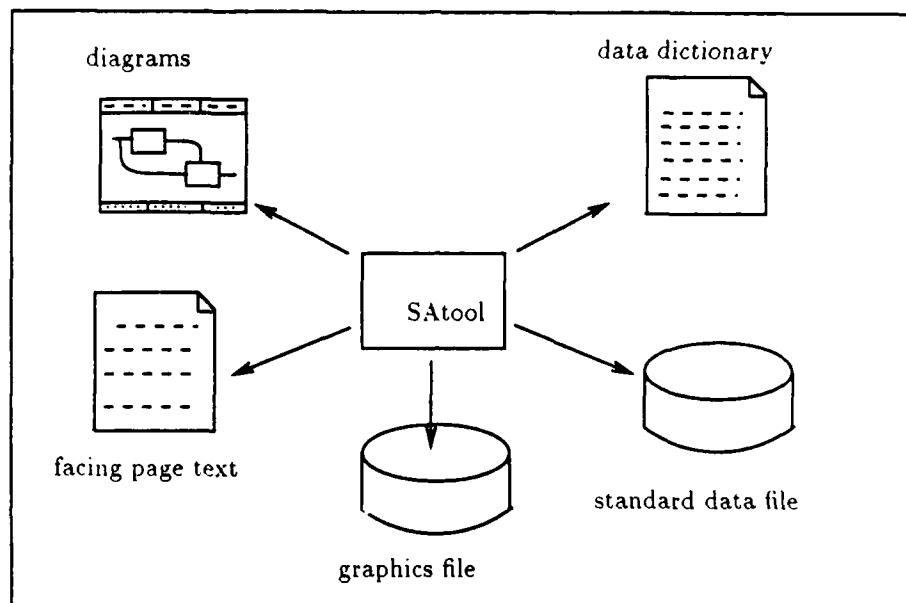


Figure 20. SAtool Products

Typical SAtool IDEF₀ Drawing Outputs

The following two drawings are typical those produced by SAtool. Figure 21 represents the so called "A minus zero" diagram, which is basically the context diagram, and Figure 22 represents the first level decomposition, the "A zero" diagram.

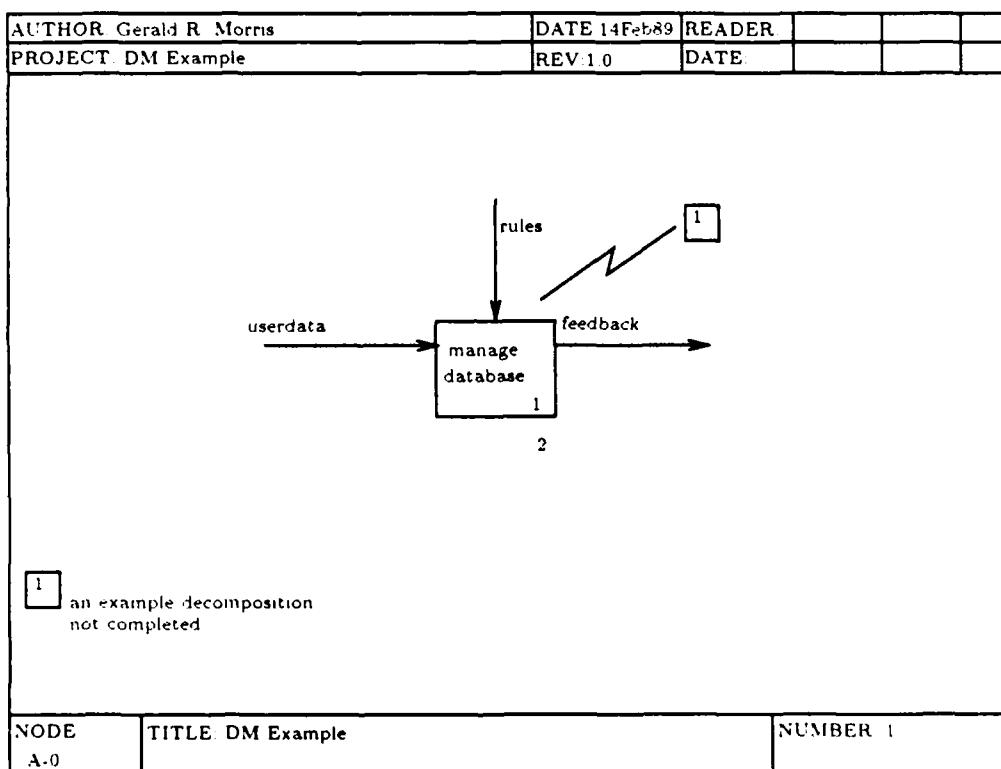


Figure 21. Typical A-0 Diagram

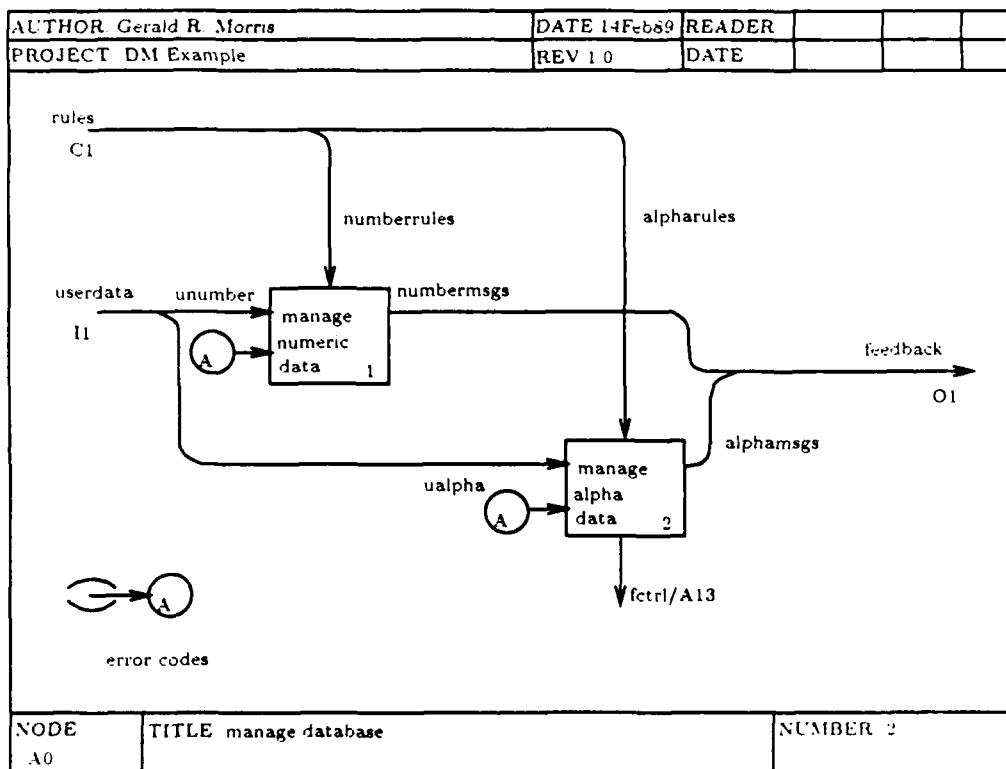


Figure 22. Typical A0 Diagram

Data Dictionary Outputs

The following two listings are representative of an activity data dictionary SAtool product, and a data element data dictionary product.

ACTIVITY Data Dictionary.

```
|  
NAME  :manage numeric  data  
TYPE  :ACTIVITY  
PROJECT  :DM Example  
NUMBER  :A1  
DESCRIPTION  :  
This activity will  
handle numbers  
INPUTS  :  
    unumber  
    errors  
OUTPUTS  :  
    numbermsgs  
CONTROLS  :  
    numberrules  
MECHANISMS  :  
  
ALIASES  :  
COMMENT  :  
PARENT ACTIVITY  :manage database  
REFERENCE:  
KNR N00028-89-0123 3.3.2.1.2a  
    REF TYPE:  
    contract  
VERSION  :1.0  
VERSION CHANGES  :  
DATE  :14Feb89  
AUTHOR  :Gerald R. Morris
```

DATA ELEMENT Data Dictionary.

NAME : unumber
TYPE : DATA ELEMENT
PROJECT :DM Example
DESCRIPTION :This is the user numeric data
DATA TYPE :integer
MIN VALUE :
MAX VALUE :
RANGE :integer'range
VALUES :
PART OF :userdata
COMPOSITION :
ALIASES :
WHERE USED :
COMMENT :
SOURCES :
DESTINATIONS :
INPUT:
 manage numeric data
REFERENCE:
 AFM 35-10 page 3 para. 2.3
 REF TYPE:
 AFM
VERSION :1.0
VERSION CHANGES :
DATE :14Feb89
AUTHOR :Gerald R. Morris

Appendix D. *Analysis Phase Data Base*

The AFIT System Development Guidelines specify the two types of items that belong in an analysis phase data dictionary - activities, and data elements (16:8). As seen in the last chapter, the SAtool output products include an ascii text data dictionary file which can be imported to the heterogeneous database via Connally's Data Manager (9). The following pages illustrate the relations for the analysis phase portion of Connally's heterogeneous database.

alias		
1	project	c12
2	aname	c25
3	aliasname	c25
4	comment	c60

ahierarchy		
1	project	c12
2	hianame	c25
3	loaname	c25

activityio		
1	project	c12
2	aname	c25
3	diname	c25
4	type	c4

bkup dirname		
1	dir_name	c100

adesc		
1	project	c12
2	aname	c25
3	line	i2
4	description	c60

dataitem		
1	project	c12
2	diname	c25
3	datatype	c25
4	low	c10
5	hi	c10
6	span	c60
7	status	c1

ahistory		
1	project	c12
2	aname	c25
3	version	c10
4	date	c8
5	author	c20
6	comment	c60

diref		
1	project	c12
2	diname	c25
3	reference	c60
4	reftype	c25

divalueaset		
1	project	c12
2	diname	c25
3	value	c15

sadact		
1	dataname	c15
2	relname	c15
3	key1	c15
4	key2	c15
5	flddesc	c4
6	entryclass	c2
7	mlfld	c1
8	numflds	c3
9	direction	c10
10	type	c10
11	delflag	c1
12	version	c15
13	line	i2

sadtdata		
1	dataname	c15
2	relname	c15
3	key1	c15
4	key2	c15
5	flddesc	c4
6	entryclass	c2
7	mlfld	c1
8	numflds	c3
9	direction	c10
10	type	c10
11	delflag	c1
12	version	c15
13	line	i2

dalias		
1	project	c12
2	diname	c25
3	aliasname	c25
4	comment	c60
5	whereused	c25

activity		
1	project	c12
2	aname	c25
3	number	c20
4	status	c1

dihistory		
1	project	c12
2	diname	c25
3	version	c10
4	date	c8
5	author	c20
6	comment	c60

ent_id_table		
1	phase	c6
2	type	c3
3	relname	c12
4	keyfld	c12

reference		
1	project	c12
2	aname	c25
3	reference	c60
4	reftype	c25

didesc		
1	project	c12
2	diname	c25
3	line	i2
4	description	c60

par_rel_tab		
1	toolname	c10
2	phase	c7
3	type	c4
4	levels	c2
5	prel_nm	c12
6	pkey_attr	c12
7	psub_attr	c12
8	crel_nm	c12
9	ckey_attr	c12
10	csub_attr	c12

sess_id_tab		
1	session_id	c12
2	project	c12
3	parent_val	c25
4	levels	c2
5	phase	c6
6	type	c4
7	owner	c20
8	tool_code	c10

monitordata		
1	time	c35
2	loginname	c50
3	action	c50

entowner_tab		
1	phase	c6
2	type	c3
3	relname	c12
4	keyfld	c12
5	owner_attr	c12

dilierarchy		
1	project	c12
2	hidiname	c25
3	lodiname	c25

tooldesc_tab		
1	code	c10
2	phase	c6
3	type	c3
4	parent_rel	c12
5	parent_attr	c12
6	child_attr	c12
7	def_table	c12
8	description	c60

Appendix E. *Typical Data Manager Session*

As mentioned in the previous chapter, Connally included a Data Manager which allows the data dictionary files to be uploaded into his heterogeneous database. Here is a typical Data Manager session:

```
ssc(1)> dm
      DATA MANAGER EXECUTION MENU

      1. Build new transaction file for execution.
      2. Use existing transaction file for execution.
      3. Exit

      ENTER CHOICE: 1

      Please enter the transaction file name: jerry

      DATA MANAGER
      TRANSACTION RECORD MENU

      TOOL SELECTION
      1. Sun SADT Editor
      2. Data Dictionary Editor

      ENTER CHOICE:[1]

      DATABASE NAME:[jerry-----]

      SESSION OWNER NAME:[DATA MANAGER-----]

      TRANSACTION INDICATOR SELECTION
      1. RETRIEVE DATA
      2. RETRIEVE DATA FOR UPDATE
      3. WRITE NEW DATA
      4. WRITE UPDATED DATA
      5. DELETE
      6. ABORT SESSION
      7. EXIT TRANSACTION MENU

      ENTER CHOICE:[3]

      SESSION FILE NAME:[dma_0-----]

      PROJECT:[DM Example--]

      TYPE SELECTION
      1. ACTIVITY
      2. DATA ITEM
      3. BOTH

      ENTER CHOICE:[3]
```

SUCCESSFUL BUILD OF TRANSACTION FILE

TYPE OF EXECUTION

1. Background (Terminal available during execution)
2. Foreground (Terminal unavailable during execution)
3. Exit

Enter Choice: 2

Data Manager now executing

Please be patient, screen may be blank for up to 30 seconds.
Possibly longer for PARENT/LEVEL transactions.

DATA MANAGER TRANSACTION RESULTS

TRANSACTION FILE NAME: jerry.ins
TOOL NAME: SADT
SESSION OWNER: DATA MANAGER
DATABASE NAME: jerry
PHASE: REQ
TYPE: BOTH
PROJECT NAME: DM Example
TYPE OF TRANSACTION: WRITE -- ALL NEW RECORDS
Updates performed using file: dma_0.dbs

ENTITIES LISTED TO BE UPDATED

manage database ACT W
feedback OBJ W
userdata OBJ W
rules OBJ W

RESULTS OF THE UPDATE

manage database ACT W successful update
feedback OBJ W successful update
userdata OBJ W successful update
rules OBJ W successful update

SUCCESSFUL UPDATE

ssc(2)>

Appendix F. Example IDEF₀ Relational Database Instance

The relational database instance shown below corresponds to the diagrams shown in Figure 7, and Figure 8. The example was constructed manually using a plain text editor, and was used to do the stepwise refinement of the relational design. Note that some of the data in the example database is in symbolic form, e.g., the locations of the various components. In addition, recall that the IDEF₀ diagrams only show drawing data. For example, the label "error codes" on the A0 diagram actually corresponds to the data element "errors" as can be determined via the **data2label** relation. In short, the essential data below is being shown for the first time.

```
act2act
parent_node child_node
1      2
1      3

act2data
node_id  data_id  icom_type
1        1        I
1        2        C
1        3        O
2        4        I
2        11       I
3        5        I
3        11       I
2        6        C
3        7        C
2        8        O
3        9        O
3        10       M

act2hist
node_id  hist_id
3        1

act2ref
node_id  ref_if
1        1
1        2
2        6
3        7

act_descr
node_id  line_no  desc_line
1        1        This is the context diagram
1        2        for the data manager analysis
2        1        This activity will
2        2        handle numbers
3        1        This activity will
3        2        handle alphanumerics
```

```

activity (part 1)
node_id  node  name          project_id  author_id
1        A0    manage database  1           1
2        A1    manage numeric data 1           1
3        A2    manage alpha data  1           1

activity (part 2)
version  date    x   y  visible_DRE sheet_id
1.0      02/14/89  1   1  -1           1
1.0      02/14/89  16  16  0           2
1.0      02/14/89  17  17  0           2

analyst
author_id  author
1          Gerald R. Morris

arrow
symbol_id  arrow_type
2          3
4          1
6          3
10         3
14         3
18         1
22         1
24         3
36         1
41         3
42         3
43         3

boundary
symbol_id  icom_code
7          I1
15         C1
23         O1

data2data
parent_data  child_data
1            4
1            5
2            6
2            7
3            8
3            9

data2label
data_id  label_id
1        1
2        2
3        3
1        4
4        5
5        6
2        7
6        8
7        9
3        10
8        11
9        12
10       13
11       14

```

```

data2ref
data_id ref_id
1      3
2      4
3      5
4      8
5      9
6      10
7      11
8      12
9      13

data2value
data_id value_id
11      1
11      2

data_descr
data_id line_no desc_line
1      1      This is the user
1      2      input data
2      1      This is the
2      2      database rules
3      1      This is the
3      2      user feedback
4      1      This is the user numeric data
5      1      The users alphanumeric data
6      1      Rules for numeric data
7      1      Rules for alphanumeric data
8      1      Feedback for numeric data
9      1      Feedback for alphanumeric data
10     1      See Flight Control Mode A13

data_elem
data_id name      project_id author_id version date
1      userdata   1          1      1.0   02/14/89
2      rules      1          1      1.0   02/14/89
3      feedback   1          1      1.0   02/14/89
4      unumber    1          1      1.0   02/14/89
5      ualpha     1          1      1.0   02/14/89
6      numberrules 1         1      1.0   02/14/89
7      alpharules 1         1      1.0   02/14/89
8      numbermsgs  1         1      1.0   02/14/89
9      alphamsgs  1         1      1.0   02/14/89
10     alphacall   1         1      1.0   02/15/89
11     errors     1         1      1.0   02/17/89

data_type
data_id type
4      integer
5      ascii
11    errcode

data_range
data_id data_range
4      integer'range
5      ascii'range

data_value
value_id value
1      bad input
2      bad output

```

```

footnote
graf_id x y
1 90 90

graphics
graf_id sheet_id
1 1
2 1

hist_call
hist_id hist_proj hist_node
1 Flight Control A13

label
label_id name x y sheet_id
1 userdata 2 2 1
2 rules 5 5 1
3 feedback 8 8 1
4 userdata 17 17 2
5 unumber 20 20 2
6 ualpha 22 22 2
7 rules 25 25 2
8 numberrules 28 28 2
9 alpharules 30 30 2
10 feedback 34 34 2
11 numbermsgs 37 37 2
12 alphamsgs 41 41 2
13 fctrl/A13 55 55 2
14 error codes 85 85 2

note
graf_id label x y
1 1 11 11

note_text
graf_id line_no text_line
1 1 an example decomposition
1 2 not completed

project
project_id name
1 DM Example

ref_type
ref_id ref_type
1 military standard
2 std operating procedure
3 military standard
4 military standard
5 military standard
6 contract
7 contract
8 AFM
9 AFM
10 AFM
11 AFM
12 AFM
13 AFM

```

```

reference
ref_id  line_no  ref_line
1       1  MIL-Std-00091
1       2  page 3 para. 7-5
2       1  System Development Guide
2       2  Draft 4
2       3  chap 2 page 7
3       1  MIL-Std-00091
3       2  page 9 para. 8-1
4       1  MIL-Std-00091
4       2  page 9 para. 8-2
5       1  MIL-Std-00091
5       2  page 9 para. 8-3
6       1  KNR M00028-89-0123 3.3.2.1.2a
7       1  KNR M00028-89-0123 3.3.2.1.2a
8       1  AFM 35-10 page 3 para. 2.3
9       1  AFM 35-10 page 3 para. 2.4
10      1  AFM 35-10 page 4 para. 3.2
11      1  AFM 35-10 page 4 para. 3.3
12      1  AFM 35-10 page 5 para. 4.5
13      1  AFM 35-10 page 5 para. 4.6

```

segment

seg_id	data_id	sheet_id	xs	ys	xe	ye
1	1	1	3	3	4	4
2	2	1	6	6	7	7
3	3	1	9	9	10	10
4	1	2	18	18	19	19
5	4	2	19	19	21	21
6	5	2	19	19	23	23
7	5	2	23	23	24	24
8	2	2	26	26	27	27
9	6	2	27	27	29	29
10	7	2	27	27	32	32
11	7	2	32	32	33	33
12	3	2	35	35	36	36
13	8	2	38	38	39	39
14	8	2	39	39	40	40
15	8	2	40	40	36	36
16	9	2	42	42	43	43
17	9	2	43	43	36	36
18	10	2	40	40	41	41
19	11	2	75	75	76	76
20	11	2	77	77	78	78
21	11	2	79	79	80	80

sheet

sheet_id	c_number
1	1
2	2

squiggle

graf_id	x1	y1	x2	y2	x3	y3	x4	y4
2	12	12	13	13	14	14	15	15

```

symbol
symbol_id seg_id sheet_id x y
 2         1       1     4   4
 4         2       1     7   7
 6         3       1    10  10
 7         4       2    18  18
10        5       2    21  21
11        6       2    19  19
13        7       2    23  23
14        7       2    24  24
15        8       2    26  26
17        8       2    27  27
18        9       2    29  29
21       10      2    32  32
22       11      2    33  33
23       12      2    35  35
24       12      2    35  35
28       13      2    39  39
30       14      2    40  40
34       16      2    43  43
35       15      2    36  36
36       18      2    40  40
37       19      2    75  75
38       19      2    76  76
39       20      2    77  77
41       21      2    79  79
42       21      2    80  80
43       19      2    76  76

to_from_all
symbol_id tfa_label
38      A
39      A
41      A

tunnel
symbol_id tunnel_type
37      -1

turn
symbol_id turn_type
11      2
13      6
17      2
21      2
28      2
30      6
34      0
35      4

```

Appendix G. *SQL Scripts*

This appendix includes the SQL scripts used to create the relational tables, perform a bulk load, and bulk erase of the relational database, show the contents of all the relational tables, and extract drawing and essential data.

Create Tables

The following SQL script creates the relations for the Ingres relational implementation of the IDEF₀ database.

```
CREATE TABLE act2act (
    parent_node integer4,
    child_node integer4);
CREATE TABLE act2data (
    node_id      integer4,
    data_id      integer4,
    icom_type   c1);
CREATE TABLE act2hist (
    node_id      integer4,
    hist_id      integer4);
CREATE TABLE act2ref (
    node_id      integer4,
    ref_id      integer4);
CREATE TABLE activity (
    node_id      integer4,
    node        c20,
    name        c25,
    project_id integer4,
    author_id   integer2,
    version     c10,
    date        c8,
    x           integer2,
    y           integer2,
    visible_DRE integer1,
    sheet_id    integer4);
CREATE TABLE act_changes (
    node_id      integer4,
    changes     c60);
CREATE TABLE act_descr (
    node_id      integer4,
    line_no     integer2,
    desc_line   c60);
CREATE TABLE alias (
    data_id      integer4,
    name        c25,
    where_used  c25,
    comment     c25);
```

```

CREATE TABLE analyst (
    author_id    integer2,
    author      c20);
CREATE TABLE arrow (
    symbol_id   integer4,
    arrow_type  integer1);
CREATE TABLE boundary (
    symbol_id   integer4,
    icom_code   c2);
CREATE TABLE data2data (
    parent_data integer4,
    child_data  integer4);
CREATE TABLE data2label (
    data_id     integer4,
    label_id    integer4);
CREATE TABLE data2ref (
    data_id     integer4,
    ref_id     integer4);
CREATE TABLE data2value (
    data_id     integer4,
    value_id   integer4);
CREATE TABLE data_changes (
    data_id     integer4,
    changes    c60);
CREATE TABLE data_descr (
    data_id     integer4,
    line_no    integer2,
    desc_line  c60);
CREATE TABLE data_elem (
    data_id     integer4,
    name       c25,
    project_id integer4,
    author_id  integer2,
    version    c10,
    date       c8);
CREATE TABLE data_range (
    data_id     integer4,
    data_range  c60);
CREATE TABLE data_type (
    data_id     integer4,
    type       c25);
CREATE TABLE data_value (
    value_id   integer4,
    value      c15);
CREATE TABLE dot (
    symbol_id   integer4,
    dot_type   integer1);
CREATE TABLE feo (
    graf_id    integer4,
    picture    c60);
CREATE TABLE footnote (
    graf_id    integer4,
    x         integer2,
    y         integer2);
CREATE TABLE graphics (
    graf_id    integer4,
    sheet_id   integer4);
CREATE TABLE hist_call (
    hist_id    integer4,
    hist_proj  c12,
    hist_node  c20);

```

```

CREATE TABLE label (
    label_id      integer4,
    name          c10,
    x              integer2,
    y              integer2,
    sheet_id      integer4);
CREATE TABLE min_max (
    data_id       integer4,
    minimum       c15,
    maximum       c15);
CREATE TABLE note (
    graf_id       integer4,
    label         c1,
    x              integer2,
    y              integer2);
CREATE TABLE note_text (
    graf_id       integer4,
    line_no       integer2,
    text_line     c60);
CREATE TABLE project (
    project_id    integer4,
    name          c12);
CREATE TABLE reference (
    ref_id        integer4,
    line_no       integer2,
    ref_line      c60);
CREATE TABLE ref_type (
    ref_id        integer4,
    ref_type      c25);
CREATE TABLE segment (
    seg_id        integer4,
    data_id       integer4,
    sheet_id      integer4
    xs             integer2,
    ys             integer2,
    xe             integer2,
    ye             integer2);
CREATE TABLE sheet (
    sheet_id      integer4,
    c_number      integer4);
CREATE TABLE squiggle (
    graf_id       integer4,
    x1            integer2,
    y1            integer2,
    x2            integer2,
    y2            integer2,
    x3            integer2,
    y3            integer2,
    x4            integer2,
    y4            integer2);
CREATE TABLE symbol (
    symbol_id     integer4,
    seg_id        integer4,
    sheet_id      integer4,
    x              integer2,
    y              integer2);
CREATE TABLE to_from_all (
    symbol_id     integer4,
    tfa_label     c1);
CREATE TABLE tunnel (
    symbol_id     integer4,
    tunnel_type   integer1);
CREATE TABLE turn (
    symbol_id     integer4,
    turn_type     integer1);

```

Load Database

The following SQL script performs a bulk load of the data in the example database into the Ingres relational implementation of the IDEF₀ database.

```
COPY TABLE act2act(parent_node=c0,child_node=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]act2act.dat";
COPY TABLE act2data(node_id=c0,data_id=c0,icom_type=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]act2data.dat";
COPY TABLE act2ref(node_id=c0,ref_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]act2ref.dat";
COPY TABLE act2hist(node_id=c0,hist_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]act2hist.dat";
COPY TABLE activity(node_id=c0,node=c0,name=c0,project_id=c0,
  author_id=c0,version=c0,date=c0,x=c0,y=c0,visible_DRE=c0,sheet_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]activity.dat";
COPY TABLE act_descr(node_id=c0,line_no=c0,desc_line=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]act_descr.dat";
COPY TABLE analyst(author_id=c0,author=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]analyst.dat";
COPY TABLE arrow(symbol_id=c0,arrow_type=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]arrow.dat";
COPY TABLE boundary(symbol_id=c0,icom_code=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]boundary.dat";
COPY TABLE data2data(parent_data=c0,child_data=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data2data.dat";
COPY TABLE data2label(data_id=c0,label_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data2label.dat";
COPY TABLE data2ref(data_id=c0,ref_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data2ref.dat";
COPY TABLE data2value(data_id=c0,value_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data2value.dat";
COPY TABLE data_descr(data_id=c0,line_no=c0,desc_line=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data_descr.dat";
COPY TABLE data_elem(data_id=c0,name=c0,project_id=c0,author_id=c0,
  version=c0,date=c0) FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data_elem.dat";
COPY TABLE data_type(data_id=c0,type=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data_type.dat";
COPY TABLE data_range(data_id=c0,data_range=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data_range.dat";
COPY TABLE data_value(value_id=c0,value=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]data_value.dat";
COPY TABLE footnote(graf_id=c0,x=c0,y=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]footnote.dat";
COPY TABLE graphics(graf_id=c0,sheet_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]graphics.dat";
COPY TABLE hist_call(hist_id=c0,hist_proj=c0,hist_node=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]hist_call.dat";
COPY TABLE label(label_id=c0,name=c0,x=c0,y=c0,sheet_id=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]label.dat";
COPY TABLE note(graf_id=c0,label=c0,x=c0,y=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]note.dat";
COPY TABLE note_text(graf_id=c0,line_no=c0,text_line=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]note_text.dat";
COPY TABLE project(project_id=c0,name=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]project.dat";
COPY TABLE reference(ref_id=c0,line_no=c0,ref_line=c0)
  FROM "DUA1:[MROTH.GMORRIS.NEWIDEFO]reference.dat";
```

```

COPY TABLE ref_type(ref_id=c0,ref_type=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]ref_type.dat";
COPY TABLE segment(seg_id=c0,data_id=c0,sheet_id=c0,xs=c0,ys=c0,xe=c0,ye=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]segment.dat";
COPY TABLE sheet(sheet_id=c0,c_number=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]sheet.dat";
COPY TABLE squiggle(graf_id=c0,x1=c0,y1=c0,x2=c0,y2=c0,
x3=c0,y3=c0,x4=c0,y4=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]squiggle.dat";
COPY TABLE symbol(symbol_id=c0,seg_id=c0,sheet_id=c0,x=c0,y=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]symbol.dat";
COPY TABLE to_from_all(symbol_id=c0,tfa_label=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]to_from_all.dat";
COPY TABLE tunnel(symbol_id=c0,tunnel_type=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]tunnel.dat";
COPY TABLE turn(symbol_id=c0,turn_type=c0)
FROM "DUA1:[MROTH.GMORRIS. NEWIDEFO]turn.dat";

```

Erase Database

The following SQL script eliminates all the data in the Ingres relational implementation of the IDEF₀ database.

```

DELETE FROM act2act;
DELETE FROM act2data;
DELETE FROM act2hist;
DELETE FROM act2ref;
DELETE FROM activity;
DELETE FROM act_changes;
DELETE FROM act_descr;
DELETE FROM alias;
DELETE FROM analyst;
DELETE FROM arrow;
DELETE FROM boundary;
DELETE FROM data2data;
DELETE FROM data2label;
DELETE FROM data2ref;
DELETE FROM data2value;
DELETE FROM data_changes;
DELETE FROM data_descr;
DELETE FROM data_elem;
DELETE FROM data_range;
DELETE FROM data_type;
DELETE FROM data_value;
DELETE FROM dot;
DELETE FROM foo;
DELETE FROM footnote;
DELETE FROM graphics;
DELETE FROM hist_call;
DELETE FROM label;
DELETE FROM min_max;
DELETE FROM note;
DELETE FROM note_text;
DELETE FROM project;
DELETE FROM reference;

```

```
DELETE FROM ref_type;
DELETE FROM segment;
DELETE FROM sheet;
DELETE FROM squiggle;
DELETE FROM symbol;
DELETE FROM to_from_all;
DELETE FROM tunnel;
DELETE FROM turn;
```

Show Database

The following SQL script shows all the data in the Ingres relational implementation of the IDEFO database.

```
SELECT * FROM act2act;
SELECT * FROM act2data;
SELECT * FROM act2hist;
SELECT * FROM act2ref;
SELECT * FROM activity;
SELECT * FROM act_changes;
SELECT * FROM act_descr;
SELECT * FROM alias;
SELECT * FROM analyst;
SELECT * FROM arrow;
SELECT * FROM boundary;
SELECT * FROM data2data;
SELECT * FROM data2label;
SELECT * FROM data2ref;
SELECT * FROM data2value;
SELECT * FROM data_changes;
SELECT * FROM data_descr;
SELECT * FROM data_elem;
SELECT * FROM data_range;
SELECT * FROM data_type;
SELECT * FROM data_value;
SELECT * FROM dot;
SELECT * FROM fec;
SELECT * FROM footnote;
SELECT * FROM graphics;
SELECT * FROM hist_call;
SELECT * FROM label;
SELECT * FROM min_max;
SELECT * FROM note;
SELECT * FROM note_text;
SELECT * FROM project;
SELECT * FROM reference;
SELECT * FROM ref_type;
SELECT * FROM segment;
SELECT * FROM sheet;
SELECT * FROM squiggle;
SELECT * FROM symbol;
SELECT * FROM to_from_all;
SELECT * FROM tunnel;
SELECT * FROM turn;
```

Extract Drawing Data

The following queries extract drawing data from the Ingres DBMS implementation of the IDEF₀ database. The first set of queries complete the drawing that was started in Chapter 7, the second set of queries extract the data to draw the A0 diagram depicted in Figure 8.

A-0 Drawing Data. The queries in this section complete the extraction of drawing data for the A-0 diagram depicted in Figure 7, that was started in Chapter 3. Recall the partially completed drawing so far consists of the sheet headers, all boxes, and all line segments.

This set of queries extract all the symbols for the ends of the segments. The symbols extracted are arrows, turns, tunnels, and to_from_all. In some instances, e.g., tunnels, there are no tuples extracted because there are none of that type of symbol. Associated with each symbol is its location, and a discriminator which indicates the type, e.g., down arrow = 1, or, in the case of to_from_all, a label.

```
select sy.x, sy.y, ar.arrow_type
from symbol sy, arrow ar
where ar.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select a.sheet_id
  from project p,activity a
  where p.project_id = a.project_id and
  a.node = "AO" and
  p.name = "DM Example");

|x      |y      |arrow_|
|-----|
|  4|  4|  3|
|  7|  7|  1|
| 10| 10|  3|
|-----|
```

```
select sy.x, sy.y, tu.turn_type
from symbol sy, turn tu
where tu.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select a.sheet_id
  from project p,activity a
  where p.project_id = a.project_id and
  a.node = "AO" and
  p.name = "DM Example");
```

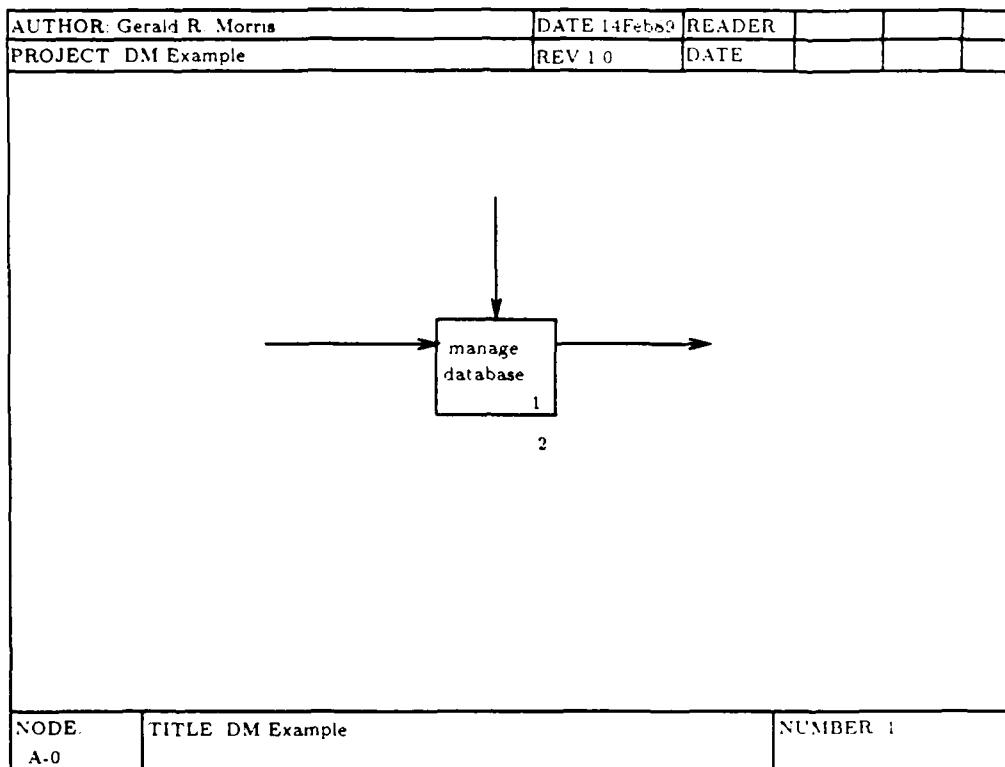


Figure 23. A-0 Diagram (partial drawing 3)

```

|x      |y      |turn_t|
|-----|-----|
|-----|-----|


select sy.x, sy.y, tu.tunnel_type
from symbol sy, tunnel tu
where tu.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select a.sheet_id
  from project p,activity a
  where p.project_id = a.project_id and
  a.node = "A0" and
  p.name = "DM Example");

|x      |y      |tunnel|
|-----|-----|
|-----|-----|

```

At this point the drawing tool adds the symbols to the drawing. The updated partial drawing that results from adding these symbols is shown in Figure 23.

This next query extracts the tuples corresponding to the labels associated with the data elements (arrows).

```
select l.x,l.y,l.name
from label l
where l.sheet_id in (
  select a.sheet_id
  from project p,activity a
  where p.project_id = a.project_id and
  a.node = "AO" and
  p.name = "DM Example");

|x      |y      |name      |
|-----|
| 2      | 2     |userdata  |
| 5      | 5     |rules     |
| 8      | 8     |feedback  |
|-----|
```

The drawing tool now adds the labels at the specified (x,y) locations resulting in the partial drawing shown in Figure 24.

These next queries extract the data associated with the other graphical constructs, i.e., squiggles, and footnotes. Note there are two set of ordered pairs associated with the footnote. The first pair is the location of the label marker, and the second is the location of the actual footnote text.

```
select s.x1, s.y1, s.x2, s.y2, s.x3, s.y3, s.x4, s.y4
from squiggle s, graphics g
where s.graf_id = g.graf_id and
g.sheet_id in (
  select a.sheet_id
  from project p,activity a
  where p.project_id = a.project_id and
  a.node = "AO" and
  p.name = "DM Example");

|x1      |y1      |x2      |y2      |x3      |y3      |x4      |y4      |
|-----|
| 12     | 12     | 13     | 13     | 14     | 14     | 15     | 15     |
|-----|
```

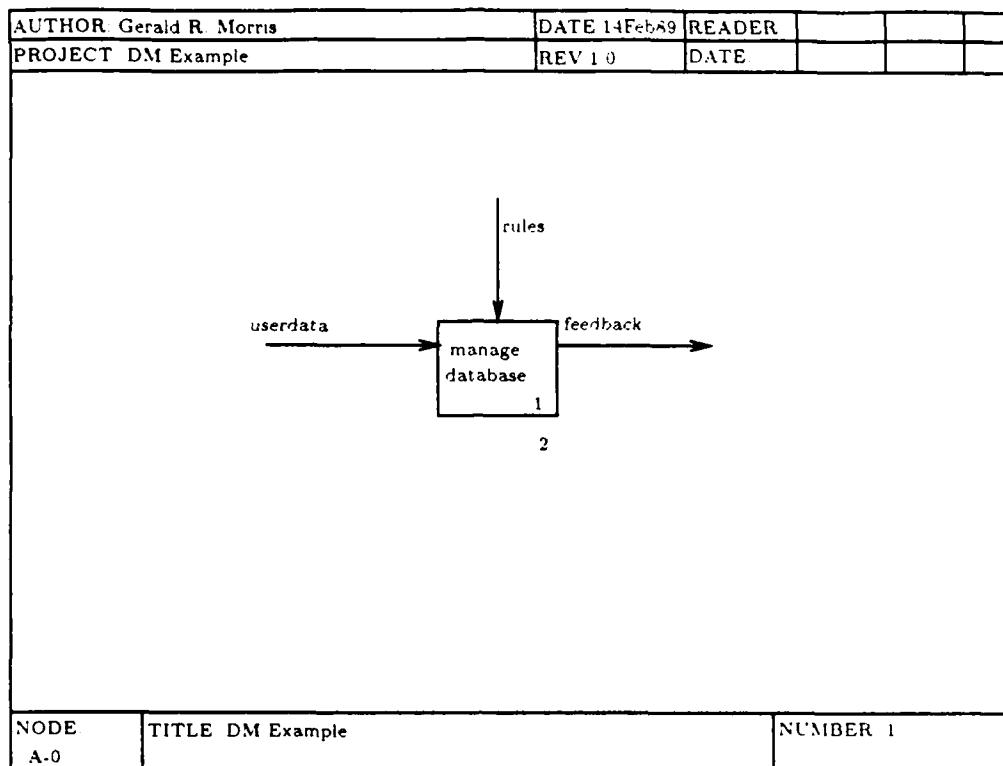


Figure 24. A-0 Diagram (partial drawing 4)

```

select xmark=n.x, ymark=n.y, n.label, xtext=f.x, ytext=f.y,
nt.line_no, nt.text_line
from note n, note_text nt, footnote f, graphics g
where n.graf_id = g.graf_id and
f.graf_id = g.graf_id and
nt.graf_id = g.graf_id and
g.sheet_id in (
  select a.sheet_id
  from project p,activity a
  where p.project_id = a.project_id and
  a.node = "A0" and
  p.name = "DM Example");

|xmark|ymark|label|xtext|ytext|line_n|text_line
|-----|
| 11| 11|1| 90| 90| 1|an example decomposition
| 11| 11|1| 90| 90| 2|not completed
|-----|

```

At this point, the drawing tool can add the footnote, and squiggle, which results in the completed diagram as previously seen in Figure 7.

A0 Drawing Data. These next queries illustrate the extraction of the drawing data associated with the A0 diagram. As before, a drawing tool might require the user to supply the name of the project being drawn, and perhaps the desired node. Accordingly, the queries shown below have "DM Example" as the project name, and "A0" as the desired node.

The first query extracts the data required to begin drawing the A0 diagram illustrated in Figure 8. As in the previous section, the table immediately following the query contains the tuples that are extracted as a result of the query.

```
select a.name, a.date, an.author, a.version, s.c_number
from activity a, analyst an, sheet s
where a.node = "A0" and
a.author_id = an.author_id and
s.sheet_id = a.sheet_id and
a.project_id in (
  select p.project_id
  from project p
  where p.name = "DM Example");



| name            | date      | author           | version | c_number |
|-----------------|-----------|------------------|---------|----------|
| manage database | 102/14/89 | Gerald R. Morris | 11.0    | 11       |


```

At this point, the drawing tool can draw the blank sheet, fill in NODE (A0), NUMBER (c-number), PROJECT ("DM Example"), TITLE (name), DATE, AUTHOR, and REV (version) for the sheet on which the A0 node is decomposed. The resulting partial drawing is shown in Figure 25.

These next queries extract all activity boxes on the A0 sheet. A join between activity and itself via act2act is needed to retrieve the desired tuples. Note the c-numbers are extracted in a separate query since non-decomposed nodes yield no tuples.

AUTHOR: Gerald R. Morris	DATE 14Feb89	READER			
PROJECT: DM Example	REV 1.0	DATE:			
NODE A0	TITLE: manage database			NUMBER: 2	

Figure 25. A0 Diagram (partial drawing 1)

```

select a.name, a.x, a.y, a.node
from activity a, act2act a2a
where a.node_id = a2a.child_node and
a2a.parent_node in (
  select a.node_id
  from activity a
  where a.node = "A0" and
  a.project_id in (
    select p.project_id
    from project p
    where p.name = "DM Example"));

```

name	x	y	node
manage alpha data	16	16	A2
manage numeric data	17	17	A1

```

select a.x, a.y, s.c_number
from activity a, act2act a2a, sheet s
where a.node_id = a2a.child_node and
a.node_id = s.node_id and
a.visible_DRE = -1 and
a2a.parent_node in (
  select a.node_id
  from activity a
  where a.node = "AO" and
  a.project_id in (
    select p.project_id
    from project p
    where p.name = "DM Example"));

```

x	y	c_number
-----	-----	-----
-----	-----	-----

Now the drawing tool can draw all the boxes at the given (x,y) locations, enter the names and node numbers into the boxes, and enter the c_number to the lower right of each activity box that has been decomposed (none in this particular case). The partial drawing resulting from this is shown in Figure 26.

The next query extracts all the line segments on this sheet. These line segments correspond to the data elements. As always, the (x,y) pairs are only symbolic in this simple example database. A "real" database would have a screen location represented in (x,y).

```

select se.xs, se.ys, se.xe, se.ye
from segment se
where se.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a.node_id = a2a.child_node and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "AO" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

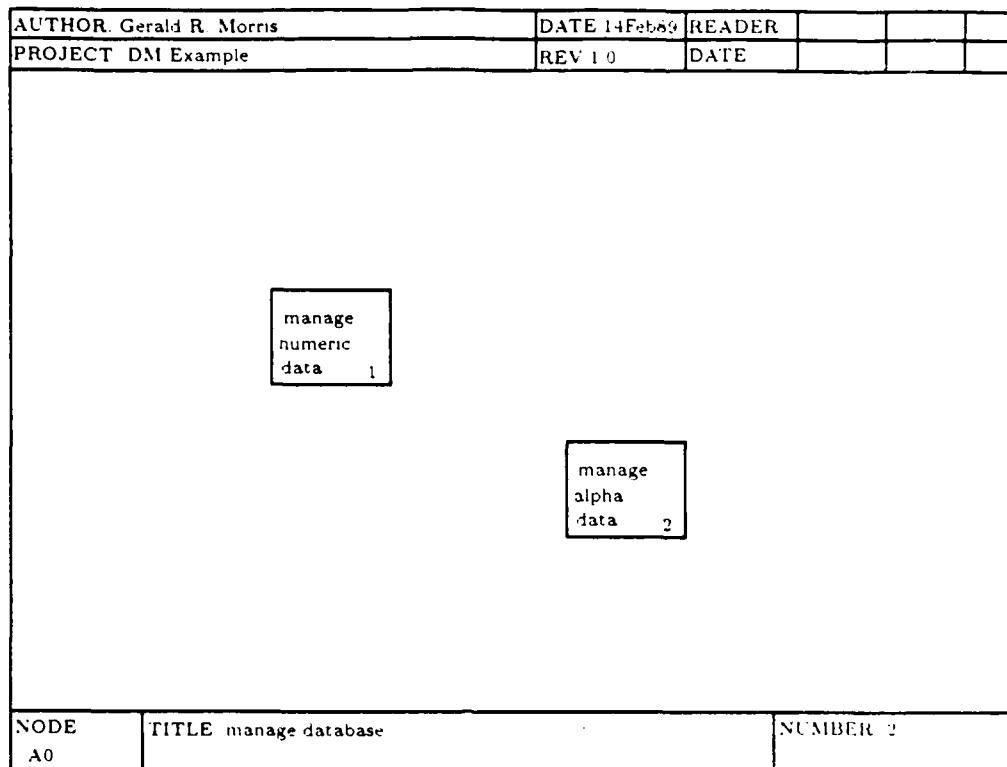


Figure 26. A0 Diagram (partial drawing 2)

18	18	19	19
19	19	21	21
19	19	23	23
23	23	24	24
26	26	27	27
27	27	29	29
27	27	32	32
32	32	33	33
35	35	36	36
38	38	39	39
39	39	40	40
40	40	36	36
40	40	41	41
42	42	43	43
43	43	36	36
75	75	76	76
77	77	78	78
79	79	80	80

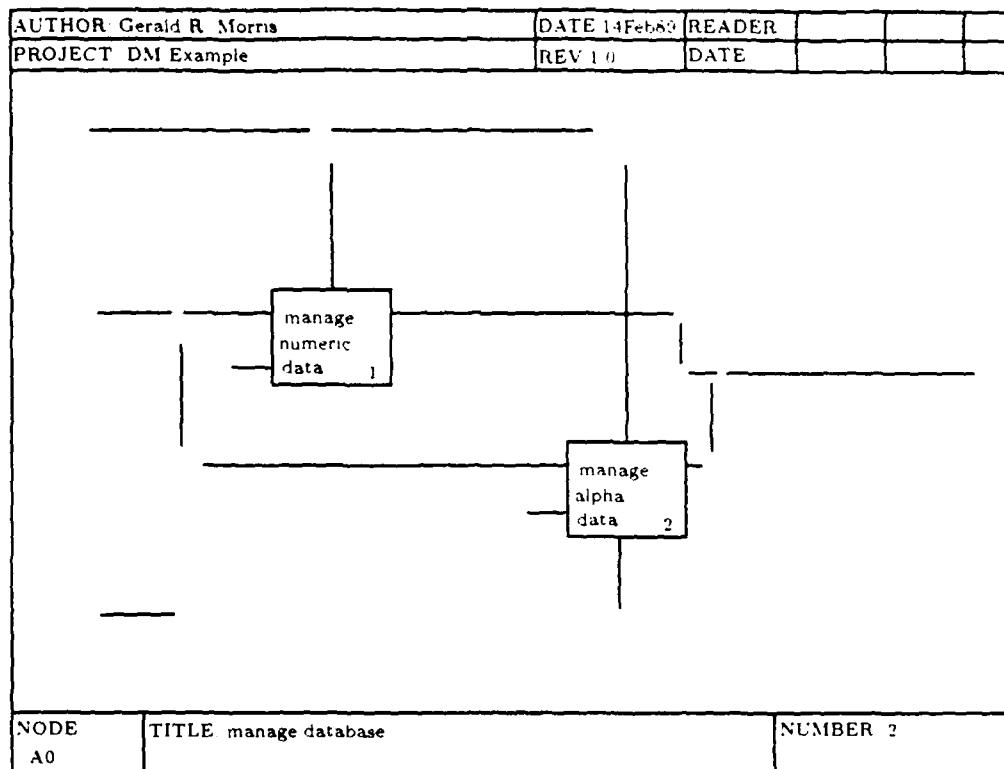


Figure 27. A0 Diagram (partial drawing 3)

Now the drawing tool can draw all the line segments at the given (x,y) locations. The partial drawing resulting from this is shown in Figure 27.

These next queries retrieve the tuples representing all the symbols at the ends of the line segments. The queries include retrieval of dots, arrows, turns, tunnels, to_from_alls, and boundaries (ICOM codes). As mentioned earlier, each symbol has an (x,y) location, and some type of discriminator or label.

```

select sy.x, sy.y, do.dot_type
from symbol sy, dot do
where do.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a.node_id = a2a.child_node and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "A0" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

x	y	dot_ty

```

select sy.x, sy.y, ar.arrow_type
from symbol sy, arrow ar
where ar.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a.node_id = a2a.child_node and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "A0" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

x	y	arrow_
21	21	3
24	24	3
29	29	1
33	33	1
35	35	3
40	40	1
78	78	3
80	80	3
76	76	3

```

select sy.x, sy.y, tu.turn_type
from symbol sy, turn tu
where tu.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a.node_id = a2a.child_node and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "A0" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

x	y	turn_t
19	19	2
23	23	6
27	27	2
32	32	2
36	36	4
39	39	2
40	40	6
43	43	0

```

select sy.x, sy.y, tu.tunnel_type
from symbol sy, tunnel tu
where tu.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a.node_id = a2a.child_node and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "A0" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

x	y	tunnel
75	75	-1

```

select sy.x, sy.y, tfa.tfa_label
from symbol sy, to_from_all tfa
where tfa.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a.node_id = a2a.child_node and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "AO" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

x	y	label
76	76	A
77	77	A
79	79	A

```

select sy.x, sy.y, bo.icon_code
from symbol sy, boundary bo
where bo.symbol_id = sy.symbol_id and
sy.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a.node_id = a2a.child_node and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "AO" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

x	y	icon_c
18	18	I1
26	26	C1
35	35	O1

Now the drawing tool can draw all the symbols for each of the line segments at the given (x,y) locations. The partial drawing resulting from this is shown in Figure 28.

This next query extracts all the labels for each of the arrows on the diagram. Associated with each label is the (x,y) location where the label is to be drawn.

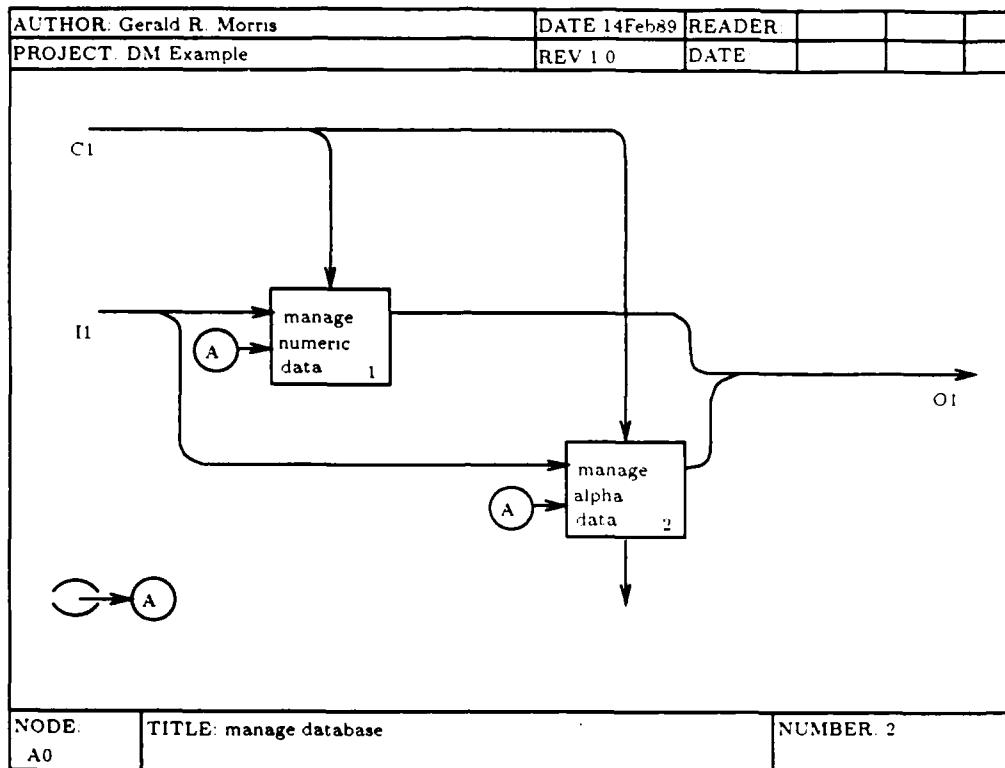


Figure 28. A0 Diagram (partial drawing 4)

```

select l.x, l.y, l.name
from label l
where l.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.sheet_id = s.sheet_id and
  a2a.child_node = a.node_id and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "A0" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));
  
```

x	y	name
17	17	userdata
20	20	unumber
22	22	alpha
25	25	rules
28	28	number:rule
30	30	alpha:rules
34	34	feedback
37	37	numbermsgs
41	41	alphamsgs
55	55	fctrl/A13
85	85	error code

Now the drawing tool can enter all the labels for each of the data elements at the given (x,y) locations. The drawing resulting from this is actually the same as the complete drawing shown in Figure 8.

These next queries extract the additional graphics entities on the drawing. In this particular instance, there are no additional graphics entities, i.e. squiggles, and footnotes.

```

select s.x1, s.y1, s.x2, s.y2, s.x3, s.y3, s.x4, s.y4
from squiggle s, graphics g
where s.graf_id = g.graf_id and
g.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.node_id = a2a.child_node and
  a.sheet_id = s.sheet_id and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "A0" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

x1	y1	x2	y2	x3	y3	x4	y4
-----	-----	-----	-----	-----	-----	-----	-----
-----	-----	-----	-----	-----	-----	-----	-----

```

select xmark=n.x, ymark=n.y, n.label, xtext=f.x, ytext=f.y,
nt.line_no, nt.text_line
from note n, note_text nt, footnote f, graphics g
where n.graf_id = g.graf_id and
f.graf_id = g.graf_id and
nt.graf_id = g.graf_id and
g.sheet_id in (
  select s.sheet_id
  from activity a, sheet s, act2act a2a
  where a.node_id = a2a.child_node and
  a.sheet_id = s.sheet_id and
  a2a.parent_node in (
    select a.node_id
    from activity a
    where a.node = "A0" and
    a.project_id in (
      select p.project_id
      from project p
      where p.name = "DM Example")));

```

```

|xmark |ymark |label |xtext |ytext |line_n|text_line
|-----|-----|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|-----|-----|

```

Extract Essential Data

The following queries extract essential data from the Ingres DBMS implementation of the IDEF₀ database. The first set of queries extract the data for a typical activity data dictionary. The second set of queries extract the data for a typical data element data dictionary. As with the drawing data above, these queries are associated with the diagram shown in Figure 8.

Activity Data Dictionary. These next queries illustrate the extraction of the data dictionary data (essential data) associated with a typical activity (A1 in the example).

```

SELECT a.name,a.node,a.version,a.date,an.author
FROM activity a,analyst an,project p
WHERE an.author_id = a.author_id and
a.project_id = p.project_id and
a.node = "A1" and
p.name = "DM Example";

```

```

|name          |node          |version |date        |author          |
|-----|-----|-----|-----|-----|
|manage numeric data |A1          |1.0     |102/14/89|Gerald R. Morris |
|-----|-----|-----|-----|-----|

```

```

SELECT c.changes /* Get the changes */
FROM activity a,act_changes c,project p
WHERE c.node_id = a.node_id and
a.project_id = p.project_id and
a.node = "A1" and
p.name = "DM Example";

|changes
|-----
|-----|
```

```

SELECT ad.line_no,ad.desc_line /* Get the description */
FROM activity a,project p,act_descr ad
WHERE a.node_id = ad.node_id and
a.project_id = p.project_id and
a.node = "A1" and
p.name = "DM Example";

|line_n|desc_line
|-----|-----|-----|
| 1|This activity will
| 2|handle numbers
|-----|
```

```

SELECT parent=a.name /* Get the parent activity */
FROM activity a,project p,act2act a2a
WHERE a.node_id = a2a.parent_node and
a2a.child_node in (
  SELECT a.node_id
  FROM activity a,project p
  WHERE a.project_id = p.project_id and
  a.node = "A1" and
  p.name = "DM Example");

|parent
|-----|
```

```

|manage database
|-----|
```

```

SELECT d.name,a2d.icom_type /* Get the data elements */
FROM activity a,data_elem d,act2data a2d,project p
WHERE a.node_id = a2d.node_id and
d.data_id = a2d.data_id and
a.project_id = p.project_id and
a.node = "A1" and
p.name = "DM Example";

|name          |icom_t|
|-----|-----|
|number        |I    |
|numberrules   |C    |
|numbermsgs   |O    |
|errors        |I    |
|-----|
```

```

SELECT rt.ref_type,r.ref_line,r.line_no /* Get the references */
FROM activity a,project p,ref_type rt,reference r,act2ref a2r
WHERE a.node_id = a2r.node_id and
r.ref_id = a2r.ref_id and
r.ref_id = rt.ref_id and
a.project_id = p.project_id and
a.node = "A1" and
p.name = "DM Example";

```

ref_type	ref_line	line_no
contract	KER M00028-89-0123 3.3.2.1.2a	1

Data Element Data Dictionary. These next queries illustrate the extraction of the data dictionary data (essential data) associated with a typical data element (unumber in the example)

```

SELECT d.name,an.author,d.version,d.date
FROM data_elem d,analyst an,project p
WHERE d.name="unumber" and
d.author_id=an.author_id and
d.project_id=p.project_id and
p.name="DM Example";

```

name	author	version	date
unumber	Gerald R. Morris	1.0	102/14/89

```

SELECT dd.line_no,dd.desc_line /* description */
FROM data_descr dd,data_elem d,project p
WHERE d.name ="unumber" and
dd.data_id = d.data_id and
p.project_id=d.project_id;

```

line_no	desc_line
1	This is the user numeric data

```

SELECT c.changes /* changes */
FROM data_changes c,data_elem d,project p
WHERE c.data_id=d.data_id and
d.name="unumber" and
p.project_id = d.project_id;

```

changes

```

SELECT a.name,a.where_used,a.comment /* aliases */
FROM alias a,data_elem d,project p
WHERE a.data_id=d.data_id and
d.name="unumber" and
p.project_id = d.project_id;

```

name	where_used	comment

```

SELECT parent=d.name /* parents */
FROM data_elem d,data2data d2d
WHERE d.data_id=d2d.parent_data and
d2d.child_data in (
  SELECT d.data_id
  FROM data_elem d,project p
  WHERE d.name="unumber" and
  d.project_id=p.project_id and
  p.name = "DM Example");

|parent          |
|-----|
|userdata        |
|-----|


SELECT children=d.name /* children */
FROM data_elem d,data2data d2d,project p
WHERE d.data_id=d2d.child_data and
d2d.parent_data in(
  SELECT d.data_id
  FROM data_elem d,project p
  WHERE d.name="unumber" and
  d.project_id=p.project_id and
  p.name = "DM Example");

|children        |
|-----|
|-----|


SELECT dt.type /* data type */
FROM data_type dt,data_elem d,project p
WHERE dt.data_id=d.data_id and
d.name="unumber" and
p.project_id=d.project_id;

|type            |
|-----|
|integer         |
|-----|


SELECT dr.data_range /* data range */
FROM data_range dr,data_elem d,project p
WHERE dr.data_id=d.data_id and
d.name="unumber" and
p.project_id=d.project_id;

|data_range      |
|-----|
|integer'range  |
|-----|


SELECT m.minimum,m.maximum /* get min/max */
FROM min_max m,data_elem d,project p
WHERE m.data_id=d.data_id and
d.name="unumber" and
p.project_id=d.project_id;

|minimum          |maximum          |
|-----|-----|
|-----|-----|

```

```

SELECT v.value /* data values */
FROM data_value v,data2value d2v,data_elem d,project p
WHERE v.value_id=d2v.value_id and
d2v.data_id=d.data_id and
d.name="unumber" and
p.project_id=d.project_id;

|value      |
|-----|
|-----|
```

```

SELECT a.name,a2d.icom_type /* sources and destinations */
FROM activity a,data_elem d,act2data a2d,project p
WHERE d.data_id=a2d.data_id and
a.node_id=a2d.node_id and
d.name="unumber" and
p.project_id=d.project_id;

|name          |icom_t|
|-----|
|manage numeric data |I    |
|-----|
```

```

SELECT rt.ref_type,r.line_no,r.ref_line /* get references */
FROM data_elem d,ref_type rt,reference r,project p,data2ref d2r
WHERE rt.ref_id=r.ref_id and
r.ref_id=d2r.ref_id and
d2r.data_id=d.data_id and
d.name="unumber" and
p.project_id=d.project_id;

|ref_type      |line_n|ref_line
|-----|
|AFM          |      1|AFM 35-10, page 3, para. 2.3
|-----|

```

Appendix H. Example IDEF₀ Nested-Relational Database Instance

The example nested-relational database shown below corresponds to the diagrams shown in Figure 7, and Figure 8. The example was constructed manually using the relational implementation as a starting point. Note that some of the data in the example database is in symbolic form, e.g., the locations of the various components, the arrow_types, etc.

The nested-relational database instance below includes the names of the relational-valued attributes in parenthesis to make things easier to understand. In addition, the table is split into three sections, activities, data elements, and sheets.

```
project

|project_name |
|-----|
|DM Example |
|-----|
(activities)
|node_id |node |name           |author           |version |date      |changes   |c_number |parent |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|1      |AO  |manage database |Gerald R. Morris |1.0    |02/14/89 |       |1        |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
(act_descr)
|line_no |descr_line           |
|-----|-----|
|1      |This is the context diagram |
|2      |for the data manager analysis |
|-----|-----|
(references)
|ref_type |
|-----|
|military standard |
|-----|
(ref_lines)
|line_no |ref_line           |
|-----|-----|
|1      |MIL-Std-00091 |
|2      |page 3 para. 7-5 |
|-----|-----|
|-----|
|std operating procedure |
|-----|
|1      |System Development Guide |
|2      |Draft 4 |
|-----|-----|
(hist_calls)
|hist_proj  |hist_node |
|-----|-----|
|-----|-----|
```

```

(data_elems)
|data_name      |icom_type |
|-----|-----|
|userdata      |I      |
|rules         |C      |
|feedback       |O      |
|-----|-----|
(children)
|node_name      |
|-----|
|manage numeric data |
|manage alpha data |
|-----|
|node_id |node |name          |author          |version |date      |changes   |c_number |parent   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|2      |A1  |manage numeric data |Gerald R. Morris |1.0    |12/14/89 |         |2        |manage database |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
(act_descr)
|line_no |descr_line          |
|-----|-----|
|1      |This activity will    |
|2      |handle numbers          |
|-----|-----|
(references)
|ref_type      |
|-----|
|contract      |
|-----|
(ref_lines)
|line_no |ref_line          |
|-----|-----|
|1      |KER E00028-89-0123 3.3.2.1.2a |
|-----|-----|
(hist_calls)
|hist_proj    |hist_node |
|-----|-----|
|-----|-----|
(data_elems)
|data_name      |icom_type |
|-----|-----|
|unumber        |I      |
|errors         |I      |
|numberrules    |C      |
|numbermsgs     |O      |
|-----|-----|
(children)
|node_name      |
|-----|
|-----|
|node_id |node |name          |author          |version |date      |changes   |c_number |parent   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|3      |A2  |manage alpha data |Gerald R. Morris |1.0    |12/14/89 |         |2        |manage database |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
(act_descr)
|line_no |descr_line          |
|-----|-----|
|1      |This activity will    |
|2      |handle alphanumerics |
|-----|-----|

```

```
(references)
|ref_type      |
|-----|
|contract      |
|-----|
(ref_lines)
|line_no |ref_line      |
|-----|-----|
|1      |KMR M00028-89-0123 3.3.2.1.2a |
|-----|-----|
(hist_calls)
|hist_proj    |hist_node |
|-----|-----|
|Flight Control |A13      |
|-----|-----|
(data_elems)
|data_name    |icom_type |
|-----|-----|
|ualpha       |I      |
|errors       |I      |
|alpharules   |C      |
|alphamsgs   |O      |
|alphacall   |M      |
|-----|-----|
(children)
|node_name      |
|-----|
|-----|
```

```

(data_elements)
|data_id |name      |author          |version |date      |changes |parent  |
|-----|-----|-----|-----|-----|-----|-----|
|1      |userdata  |Gerald R. Morris |1.0    |02/14/89 |      |      |
|-----|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no |descr_line          |
|-----|-----|
|1      |This is the user    |
|2      |input data           |
|-----|-----|
(references)
|ref_type          |
|-----|
|military standard|
|-----|
(ref_lines)
|line_no |ref_line          |
|-----|-----|
|1      |MIL-Std-00091      |
|2      |page 9 para. 8-1    |
|-----|-----|
(aliases)
|name      |where_used |comment      |
|-----|-----|-----|
|-----|-----|-----|
(min_max)
|data_type |minimum    |maximum    |
|-----|-----|-----|
|-----|-----|-----|
(range)
|data_type |range      |
|-----|-----|
|-----|-----|
(values)
|data_type |value      |
|-----|-----|
|-----|-----|
(activitees)
|node_name      |licom_type |
|-----|-----|
|manage database|I          |
|-----|-----|
(children)
|data_name      |
|-----|
|unumber        |
|ualpha         |
|-----|
|data_id |name      |author          |version |date      |changes |parent  |
|-----|-----|-----|-----|-----|-----|-----|
|2      |rules      |Gerald R. Morris |1.0    |02/14/89 |      |      |
|-----|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no |descr_line          |
|-----|-----|
|1      |This is the          |
|2      |database rules       |
|-----|-----|

```

```

(refrences)
|ref_type |
|-----|
|military standard |
|-----|
(ref_lines)
|line_no |ref_line |
|-----|-----|
|1 |MIL-Std-00091 |
|2 |page 9 para. 8-2 |
|-----|-----|
(aliases)
|name |where_used |comment |
|-----|-----|-----|
|-----|-----|-----|
(min_max)
|data_type |minimum |maximum |
|-----|-----|-----|
|-----|-----|-----|
(range)
|data_type |range |
|-----|-----|
|-----|-----|
(values)
|data_type |value |
|-----|-----|
|-----|-----|
(activitees)
|node_name |icom_type |
|-----|-----|
|manage database |C |
|-----|-----|
(children)
|data_name |
|-----|
|numberrules |
|alpharules |
|-----|
|data_id |name |author |version |date |changes |parent |
|-----|-----|-----|-----|-----|-----|-----|
|3 |feedback |Gerald R. Morris |1.0 |02/14/89 | | |
|-----|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no |descr_line |
|-----|-----|
|1 |This is the |
|2 |user feedback |
|-----|-----|
(refrences)
|ref_type |
|-----|
|military standard |
|-----|
(ref_lines)
|line_no |ref_line |
|-----|-----|
|1 |MIL-Std-00091 |
|2 |page 9 para. 8-3 |
|-----|-----|
(aliases)
|name |where_used |comment |
|-----|-----|-----|
|-----|-----|-----|

```

```

(min_max)
|data_type |minimum      |maximum      |
|-----|-----|-----|
|-----|-----|-----|
(range)
|data_type |range      |
|-----|-----|
|-----|-----|
(values)
|data_type |value      |
|-----|-----|
|-----|-----|
(activitees)
|node_name      |icom_type |
|-----|-----|
|manage database |0      |
|-----|-----|
(children)
|data_name      |
|-----|
|numbermsgs      |
|alphamsgs      |
|-----|
|data_id |name      |author      |version |date      |changes |parent  |
|-----|-----|-----|-----|-----|-----|-----|
|4      |unumber   |Gerald R. Morris |1.0    |02/14/89 |        |userdata |
|-----|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no |descr_line      |
|-----|-----|
|1      |This is the user numeric data |
|-----|-----|
(references)
|ref_type      |
|-----|
|AFM          |
|-----|
(ref_lines)
|line_no |ref_line      |
|-----|-----|
|1      |AFM 35-10 page 3 para. 2.3 |
|-----|-----|
(aliases)
|name      |where_used      |comment      |
|-----|-----|-----|
|-----|-----|-----|
(min_max)
|data_type |minimum      |maximum      |
|-----|-----|-----|
|-----|-----|-----|
(range)
|data_type |range      |
|-----|-----|
|integer  |integer'range |
|-----|-----|
(values)
|data_type |value      |
|-----|-----|
|-----|-----|
(activitees)
|node_name      |icom_type |
|-----|-----|
|manage numeric data |I      |
|-----|-----|

```

```

(children)
  |data_name      |
  |-----|
  |-----|
  |data_id |name      |author          |version |date      |changes |parent   |
  |-----|-----|-----|-----|-----|-----|-----|
  |5      |alpha      |Gerald R. Morris |1.0     |02/14/89 |       |userdata |
  |-----|-----|-----|-----|-----|-----|-----|
(data_descr)
  |line_no |descr_line          |
  |-----|
  |1       |The users alphanumeric data |
  |-----|
(references)
  |ref_type          |
  |-----|
  |AFM              |
  |-----|
(ref_lines)
  |line_no |ref_line          |
  |-----|
  |1       |AFM 35-10 page 3 para. 2.4 |
  |-----|
(aliases)
  |name    |where_used |comment      |
  |-----|-----|-----|
  |-----|-----|-----|
(min_max)
  |data_type |minimum    |maximum    |
  |-----|-----|-----|
  |-----|-----|-----|
(range)
  |data_type |range          |
  |-----|
  |ascii    |ascii'range |
  |-----|
(values)
  |data_type |value          |
  |-----|
  |-----|
(activitees)
  |node_name      |icom_type |
  |-----|
  |manage alpha data |I      |
  |-----|
(children)
  |data_name      |
  |-----|
  |-----|
  |data_id |name      |author          |version |date      |changes |parent   |
  |-----|-----|-----|-----|-----|-----|-----|
  |6      |numberrules|Gerald R. Morris |1.0     |02/14/89 |       |rules    |
  |-----|-----|-----|-----|-----|-----|-----|
(data_descr)
  |line_no |descr_line          |
  |-----|
  |1       |Rules for numeric data |
  |-----|

```

```

(references)
|ref_type
|-----
|AFM
|-----
(ref_lines)
|line_no |ref_line
|-----|-----
|1      |AFM 35-10 page 4 para. 3.2
|-----|-----
(aliases)
|name   |where_used   |comment
|-----|-----|-----|
|-----|-----|-----|
(min_max)
|data_type |minimum      |maximum      |
|-----|-----|-----|
|-----|-----|-----|
(range)
|data_type |range
|-----|-----|
|-----|-----|
(values)
|data_type |value
|-----|-----|
|-----|-----|
(activitees)
|node_name      |icom_type
|-----|-----|
|manage numeric data |C
|-----|-----|
(children)
|data_name
|-----|
|-----|
|data_id |name      |author          |version |date      |changes |parent
|-----|-----|-----|-----|-----|-----|-----|
|7      |alpharules |Gerald R. Morris |1.0    |02/14/89 |        |rules
|-----|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no |descr_line
|-----|-----|
|1      |Rules for alphanumeric data
|-----|-----|
(references)
|ref_type
|-----
|AFM
|-----
(ref_lines)
|line_no |ref_line
|-----|-----|
|1      |AFM 35-10 page 4 para. 3.3
|-----|-----
(aliases)
|name   |where_used   |comment
|-----|-----|-----|
|-----|-----|-----|
(min_max)
|data_type |minimum      |maximum      |
|-----|-----|-----|
|-----|-----|-----|

```

```

(range)
|data_type |range      |
|-----|-----|
|-----|-----|
(values)
|data_type |value      |
|-----|-----|
|-----|-----|
(activitees)
|node_name      |icom_type |
|-----|-----|
|manage alpha data |C      |
|-----|-----|
(children)
|data_name      |
|-----|
|-----|
|data_id |name      |author      |version |date      |changes |parent   |
|-----|-----|-----|-----|-----|-----|-----|
|8     |numbermsgs |Gerald R. Morris |1.0    |02/14/89 |feedback |
|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no |descr_line      |
|-----|-----|
|1      |Feedback for numeric data |
|-----|-----|
(references)
|ref_type      |
|-----|
|AFM          |
|-----|
(ref_lines)
|line_no |ref_line      |
|-----|
|1      |AFM 35-10 page 5 para. 4.5 |
|-----|
(aliases)
|name      |where_used      |comment      |
|-----|-----|-----|
|-----|-----|-----|
(min_max)
|data_type |minimum      |maximum      |
|-----|-----|-----|
|-----|-----|
(range)
|data_type |range      |
|-----|-----|
|-----|-----|
(values)
|data_type |value      |
|-----|-----|
|-----|-----|
(activitees)
|node_name      |icom_type |
|-----|-----|
|manage numeric data |0      |
|-----|-----|
(children)
|data_name      |
|-----|
|-----|
|data_id |name      |author      |version |date      |changes |parent   |
|-----|-----|-----|-----|-----|-----|-----|
|9     |alphamsgs |Gerald R. Morris |1.0    |02/14/89 |feedback |
|-----|-----|-----|-----|-----|-----|

```

```

(data_descr)
  |line_no |descr_line
  |-----|-----
  |1      |Feedback for alphanumeric data
  |-----|-----
  (references)
    |ref_type
    |-----
    |AFM
    |-----|
  (ref_lines)
    |line_no |ref_line
    |-----|-----
    |1      |AFM 35-10 page 5 para. 4.6
    |-----|-----
  (aliases)
    |name      |where_used      |comment
    |-----|-----|-----|
    |-----|-----|-----|
  (min_max)
    |data_type |minimum      |maximum
    |-----|-----|-----|
    |-----|-----|-----|
  (range)
    |data_type |range
    |-----|-----|
    |-----|-----|
  (values)
    |data_type |value
    |-----|-----|
    |-----|-----|
  (activitees)
    |node_name      |icom_type
    |-----|-----|
    |manage alpha data |0
    |-----|-----|
  (children)
    |data_name
    |-----|
    |-----|
    |data_id |name      |author      |version |date      |changes |parent
    |-----|-----|-----|-----|-----|-----|-----|
    |10    |alphacall |Gerald R. Morris |1.0    |02/15/89 |      |      |
    |-----|-----|-----|-----|-----|-----|-----|
  (data_descr)
    |line_no |descr_line
    |-----|-----|
    |1      |See Flight Control Node A13
    |-----|-----|
  (references)
    |ref_type
    |-----|-----|
    |-----|-----|
  (ref_lines)
    |line_no |ref_line
    |-----|-----|
    |-----|-----|
  (aliases)
    |name      |where_used      |comment
    |-----|-----|-----|
    |-----|-----|-----|
  (min_max)
    |data_type |minimum      |maximum
    |-----|-----|-----|
    |-----|-----|-----|

```

```

(range)
  |data_type |range      |
  |-----|-----|
  |-----|-----|
(values)
  |data_type |value      |
  |-----|-----|
  |-----|-----|
(activitees)
  |node_name      |icom_type |
  |-----|-----|
  |manage alpha data |M      |
  |-----|-----|
(children)
  |data_name      |
  |-----|
  |-----|
  |data_id |name      |author      |version |date      |changes |parent  |
  |-----|-----|-----|-----|-----|-----|-----|
  |11    |errors    |Gerald R. Morris |1.0    |02/17/89 |      |      |
  |-----|-----|-----|-----|-----|-----|-----|
(data_descr)
  |line_no |descr_line      |
  |-----|-----|
  |-----|-----|
(references)
  |ref_type      |
  |-----|
  |-----|
(ref_lines)
  |line_no |ref_line      |
  |-----|-----|
  |-----|-----|
(aliases)
  |name      |where_used      |comment      |
  |-----|-----|-----|
  |-----|-----|-----|
(min_max)
  |data_type |minimum      |maximum      |
  |-----|-----|-----|
  |-----|-----|-----|
(range)
  |data_type |range      |
  |-----|-----|
  |-----|-----|
(values)
  |data_type |value      |
  |-----|-----|
  |errcode  |bad input    |
  |errcode  |bad output    |
  |-----|-----|
(activitees)
  |node_name      |icom_type |
  |-----|-----|
  |manage numeric data |I      |
  |manage alpha data  |I      |
  |-----|-----|
(children)
  |data_name      |
  |-----|
  |-----|

```

```

(sheets)
|c_number|node  |name          |author          |version |date   |
|-----|-----|-----|-----|-----|-----|
|1      |A-0   |DM Example  |Gerald R. Morris |1.0    |02/14/89|
|-----|-----|-----|-----|-----|-----|
(boxes)
|node |name          |x |y |visible_DRE |
|-----|-----|---|---|-----|
|A0   |manage database |1 |1 |2
|-----|-----|---|---|-----|
(segments)
|data_id |
|-----|
|1
|-----|
(location)
|xs |ys |xe |ye |
|---|---|---|---|
|3  |3  |4  |4  |
|---|---|---|---|
(symbols)
|x |y |type_symbol |symbol_type |
|---|---|-----|-----|
|4  |4  |arrow      |right_arrow |
|---|---|-----|-----|
|-----|
|2
|-----|
|---|---|---|---|
|6  |6  |7  |7  |
|---|---|---|---|
|---|---|-----|-----|
|7  |7  |arrow      |down_arrow |
|---|---|-----|-----|
|-----|
|3
|-----|
|---|---|---|---|
|9  |9  |10 |10 |
|---|---|---|---|
|---|---|-----|-----|
|10 |10 |arrow      |right_arrow |
|---|---|-----|-----|
(squiggles)
|x1 |y1 |x2 |y2 |x3 |y3 |x4 |y4 |
|---|---|---|---|---|---|---|---|
|12 |12 |13 |13 |14 |14 |15 |15 |
|---|---|---|---|---|---|---|---|
(meta_notes)
|label |x |y |
|-----|---|---|
|-----|---|---|
(note_text)
|line_no |text_line
|-----|-----|
|-----|-----|

```

```

(foot_notes)
|label |xm |ym |xn |yn |
|-----|---|---|---|---|
|1 |11 |11 |90 |90 |
|-----|---|---|---|---|
(note_text)
|line_no |text_line |
|-----|-----|
|1 |an example decomposition |
|2 |not completed |
|-----|-----|
(feos)
|label |x |y |picture |
|-----|---|---|-----|
|-----|---|---|-----|
(labels)
|data_id |name |x |y | | |
|---|---|---|---|---|---|
|1 |userdata |2 |2 |
|2 |rules |5 |5 |
|3 |feedback |8 |8 |
|-----|-----|---|---|
|c_number |node |name |author |version |date |
|-----|-----|-----|-----|-----|-----|
|2 |A0 |manage database |Gerald R. Morris |1.0 |02/14/89 |
|-----|-----|-----|-----|-----|-----|
(boxes)
|node |name |x |y |visible_DRE |
|-----|-----|---|---|-----|
|A1 |manage numeric data |16 |16 | -1 |
|A2 |manage alpha data |17 |17 | -1 |
|-----|-----|---|---|-----|
(segments)
|data_id |
|-----|
|1 |
|-----|
(location)
|xs |ys |xe |ye |
|---|---|---|---|
|18 |18 |19 |19 |
|---|---|---|---|
(symbols)
|x |y |type_symbol |symbol_type |
|---|---|-----|-----|
|18 |18 |boundary |I1 |
|---|---|-----|-----|
|-----|
|4 |
|-----|
|---|---|---|---|
|19 |19 |21 |21 |
|---|---|---|---|
|-----|
|21 |21 |arrow |right-arrow |
|---|---|-----|-----|
|-----|
|5 |
|-----|
|---|---|---|---|
|19 |19 |23 |23 |
|---|---|---|---|
|-----|
|19 |19 |turn |right-down |
|---|---|-----|-----|

```

```

|---|---|---|---|
|23 |23 |24 |24 |
|---|---|---|---|
|---|---|-----|-----|
|23 |23 |turn      |down-right   |
|24 |24 |arrow     |right-arrow  |
|---|---|-----|-----|
|-----|
|2      |
|-----|
|---|---|---|---|
|26 |26 |27 |27 |
|---|---|---|---|
|---|---|-----|-----|
|26 |26 |boundary |C1          |
|27 |27 |turn      |right-down  |
|---|---|-----|-----|
|-----|
|6      |
|-----|
|---|---|---|---|
|27 |27 |29 |29 |
|---|---|---|---|
|---|---|-----|-----|
|29 |29 |arrow     |down-arrow   |
|---|---|-----|-----|
|-----|
|7      |
|-----|
|---|---|---|---|
|27 |27 |32 |32 |
|---|---|---|---|
|---|---|-----|-----|
|32 |32 |turn      |right-down  |
|---|---|-----|-----|
|---|---|---|---|
|32 |32 |33 |33 |
|---|---|---|---|
|---|---|-----|-----|
|33 |33 |arrow     |down-arrow   |
|---|---|-----|-----|
|-----|
|3      |
|-----|
|---|---|---|---|
|35 |35 |36 |36 |
|---|---|---|---|
|---|---|-----|-----|
|35 |35 |boundary |01          |
|35 |35 |arrow     |right-arrow  |
|---|---|-----|-----|

```

```

|-----|
|8      |
|-----|
|---|---|---|---|
|38 |38 |39 |39 |
|---|---|---|---|
|---|---|-----|
|39 |39 |turn      |right-down  |
|---|---|-----|
|---|---|---|---|
|39 |39 |40 |40 |
|---|---|---|---|
|---|---|-----|
|40 |40 |turn      |down-right  |
|---|---|-----|
|---|---|---|---|
|40 |40 |36 |36 |
|---|---|---|---|
|---|---|-----|
|---|---|-----|
|-----|
|9      |
|-----|
|---|---|---|---|
|42 |42 |43 |43 |
|---|---|---|---|
|---|---|-----|
|43 |43 |turn      |right-up    |
|---|---|-----|
|---|---|---|---|
|43 |43 |36 |36 |
|---|---|---|---|
|---|---|-----|
|36 |36 |turn      |up-right    |
|---|---|-----|
|-----|
|10     |
|-----|
|---|---|---|---|
|40 |40 |41 |41 |
|---|---|---|---|
|---|---|-----|
|40 |40 |arrow    |down-arrow  |
|---|---|-----|
|-----|
|11     |
|-----|
|---|---|---|---|
|75 |75 |76 |76 |
|---|---|---|---|
|---|---|-----|
|75 |75 |tunnel    |hidden-source |
|76 |76 |to-from-all |A
|76 |76 |arrow     |right-arrow  |
|---|---|-----|
|---|---|---|---|
|77 |77 |78 |78 |
|---|---|---|---|
|---|---|-----|
|77 |77 |to-from-all |A
|78 |78 |arrow     |right-arrow  |
|---|---|-----|
|---|---|---|---|
|79 |79 |80 |80 |
|---|---|---|---|

```

```

|---|---|-----|-----|
|79 |79 |to-from-all |▲
|80 |80 |arrow      |right-arrow  |
|---|---|-----|-----|
(squiggles)
|x1 |y1 |x2 |y2 |x3 |y3 |x4 |y4 |
|---|---|---|---|---|---|---|---|
|---|---|---|---|---|---|---|---|
(meta_notes)
|label |x |y |
|-----|---|---|
|-----|---|---|
(note_text)
|line_no |text_line      |
|-----|-----|
|-----|-----|
(foot_notes)
|label |xm |ym |xn |yn |
|-----|---|---|---|---|
|-----|---|---|---|---|
(note_text)
|line_no |text_line      |
|-----|-----|
|-----|-----|
(feos)
|label |x |y |picture      |
|-----|---|---|-----|
|-----|---|---|-----|
(labels)
|data_id |name      |x |y |
|-----|-----|---|---|
|1      |userdata  |17 |17 |
|4      |unumber   |20 |20 |
|5      |ualpha    |22 |22 |
|2      |rules     |25 |25 |
|6      |numberrules |28 |28 |
|7      |alpharules |30 |30 |
|3      |feedback   |34 |34 |
|8      |numbermsgs |37 |37 |
|9      |alphamsgs  |41 |41 |
|10     |fctrl/A13  |55 |55 |
|11     |error code |85 |85 |
|-----|-----|---|---|

```

Appendix I. *SQL/NF Scripts*

This appendix includes the SQL/NF scripts that create the nested-relational database, perform a bulk load, and bulk erase of the database, show the contents of the nested-relational database, and extract data from the database.

Create Tables

The following SQL/NF script creates the schema for the nested-relational implementation of the IDEF₀ database.

```
SCHEME
  TABLE DESCRIPT
    ITEM line_no INTEGER 2
    ITEM desc_line CHARACTER 60
  TABLE REF
    ITEM ref_type CHARACTER 25
    ITEM (TABLE ref_lines
      ITEM line_no INTEGER 2
      ITEM ref_line CHARACTER 60)
  TABLE NOTETEXT
    ITEM line_no INTEGER 2
    ITEM text_line CHARACTER 60
SCHEMA
  TABLE PROJECT
    ITEM project_name CHARACTER 12 UNIQUE
    ITEM (TABLE activities
      ITEM node_id INTEGER 4 UNIQUE NOT NULL
      ITEM node CHARACTER 20
      ITEM name CHARACTER 25
      ITEM author CHARACTER 20
      ITEM version CHARACTER 10
      ITEM date CHARACTER 8
      ITEM changes CHARACTER 60
      ITEM c_number INTEGER 4 REFERENCES PROJECT.sheets.c_number
      ITEM parent CHARACTER 25 REFERENCES PROJECT.activities.name
      ITEM (TABLE act_descr DESCRIPT)
      ITEM (TABLE references REF) /* confusing name choice here! */
      ITEM (TABLE hist_calls
        ITEM hist_proj CHARACTER 12
        ITEM hist_node CHARACTER 20)
      ITEM (TABLE data_elems
        ITEM data_name CHARACTER 25 REFERENCES PROJECT.data_elements.name
        ITEM icom_type CHARACTER 1)
      ITEM (TABLE children
        ITEM node_name CHARACTER 25 REFERENCES PROJECT.activities.name))
    ITEM (TABLE data_elements
      ITEM data_id INTEGER 4 UNIQUE NOT NULL
      ITEM name CHARACTER 25
      ITEM author CHARACTER 20
      ITEM version CHARACTER 10
```

```

ITEM date CHARACTER 8
ITEM changes CHARACTER 60
ITEM parent CHARACTER 25 REFERENCES PROJECT.data_elements.name
ITEM (TABLE data_descr DESCRIPT)
ITEM (TABLE references REF)
ITEM (TABLE aliases
      ITEM name CHARACTER 25
      ITEM where_used CHARACTER 25
      ITEM comment CHARACTER 25)
ITEM (TABLE min_max
      ITEM data_type CHARACTER 25
      ITEM minimum CHARACTER 15
      ITEM maximum CHARACTER 15)
ITEM (TABLE range
      ITEM data_type CHARACTER 25
      ITEM range CHARACTER 60)
ITEM (TABLE values
      ITEM data_type CHARACTER 25
      ITEM value CHARACTER 15)
ITEM (TABLE activitees
      ITEM node_name CHARACTER 25 REFERENCES PROJECT.activities.name)
ITEM icom_type CHARACTER 1)
ITEM (TABLE children
      ITEM data_name CHARACTER 25 REFERENCES PROJECT.data_elements.name)
ITEM (TABLE sheets
      ITEM c_number INTEGER 4 UNIQUE NOT NULL
      ITEM node CHARACTER 20 REFERENCES PROJECT.activities.node
      ITEM name CHARACTER 25 REFERENCES PROJECT.activities.name
      ITEM author CHARACTER 20 REFERENCES PROJECT.activities.author
      ITEM version CHARACTER 10 REFERENCES PROJECT.activities.version
      ITEM date CHARACTER 8 REFERENCES PROJECT.activities.date
      ITEM (TABLE boxes
            ITEM node CHARACTER 20 REFERENCES PROJECT.activities.node
            ITEM name CHARACTER 25 REFERENCES PROJECT.activities.name
            ITEM x INTEGER 2
            ITEM y INTEGER 2
            ITEM visible_dre INTEGER 2)
      ITEM (TABLE segments
            ITEM data_id INTEGER 4 REFERENCES PROJECT.data_elements.data_id
            ITEM (TABLE location
                  ITEM xs INTEGER 2
                  ITEM ys INTEGER 2
                  ITEM xe INTEGER 2
                  ITEM ye INTEGER 2)
            ITEM (TABLE symbols
                  ITEM x INTEGER 2
                  ITEM y INTEGER 2
                  ITEM type_symbol INTEGER 2
                  ITEM symbol_type INTEGER 2))
      ITEM (TABLE squiggles
            ITEM x1 INTEGER 2
            ITEM y1 INTEGER 2
            ITEM x2 INTEGER 2
            ITEM y2 INTEGER 2
            ITEM x3 INTEGER 2
            ITEM y3 INTEGER 2
            ITEM x4 INTEGER 2
            ITEM y4 INTEGER 2)
      ITEM (TABLE meta_notes
            ITEM label CHARACTER 1
            ITEM x INTEGER 2
            ITEM y INTEGER 2
            ITEM (TABLE note_text NOTETEXT))
      ITEM (TABLE foot_notes
            ITEM label CHARACTER 1

```

```

ITEM xm INTEGER 2
ITEM ym INTEGER 2
ITEM xn INTEGER 2
ITEM yn INTEGER 2
ITEM (TABLE note_text NOTETEXT))
ITEM (TABLE feos
      ITEM label CHARACTER 1
      ITEM x INTEGER 2
      ITEM y INTEGER 2
      ITEM picture CHARACTER 60)
ITEM (TABLE labels
      ITEM data_id INTEGER 4 REFERENCES PROJECT.data_elements.data_id
      ITEM name CHARACTER 10
      ITEM x INTEGER 2
      ITEM y INTEGER 2)

```

Load Database

The following SQL/NF script does a bulk load of the data in the example database into the nested-relational implementation of the IDEF₀ database. Note that Roth's SQL/NF data manipulation language does not actually contain the DML command which allows a bulk load. The syntax shown below is based on the syntax associated with the Ingres SQL. Note that a percent sign (%) is used to delimit nested relations (as denoted by the =c0% format), and a comma or carriage return are used to delimit atomic values (as denoted by the =c0 format).

```

COPY TABLE project
(
  project_name=c0,
  activities(node_id=c0,node=c0,name=c0,author=c0,
             version=c0,date=c0,changes=c0,c_number=c0,parent=c0,
             act_descr(line_no=c0,desc_line=c0)=c0%,
             references(ref_type=c0,ref_lines(line_no=c0,ref_line=c0)=c0%)=c0%,
             hist_calls(hist_proj=c0,hist_node=c0)=c0%,
             data_elems(data_name=c0,icom_type=c0)=c0%,
             children(node_name=c0)=c0%
  )=c0%,
  data_elements(data_id=c0,name=c0,author=c0,
                 version=c0,date=c0,changes=c0,parent=c0,
                 data_descr(line_no=c0,desc_line=c0)=c0%,
                 references(ref_type=c0,ref_lines(line_no=c0,ref_line=c0)=c0%)=c0%,
                 aliases(name=c0,where_used=c0,comment=c0)=c0%,
                 min_max(data_type=c0,minum=c0,maximum=c0)=c0%,
                 range(data_type=c0,range=c0)=c0%,
                 values(data_type=c0,value=c0)=c0%,
                 children(data_name=c0)=c0%
  )=c0%,
  sheets(c_number=c0,node=c0,name=c0,author=c0,version=c0,date=c0,
         boxes(node=c0,name=c0,x=c0,y=c0,visible_dre=c0)=c0%,
         segments(data_id=c0,location(xs=c0,ys=c0,x=c0,y=c0)=c0%,
                  symbols(x=c0,y=c0,symbol_type=c0,type_symbol=c0)=c0%)=c0%,
                  squiggles(x1=c0,y1=c0,x2=c0,y2=c0,x3=c0,y3=c0,x4=c0,y4=c0)=c0%,

```

```
meta_notes(label=c0,x=c0,y=c0,note_text(line_no=c0,text_line=c0)=c0%)=c0%,
foot_notes(label=c0,xm=c0,ym=c0,xn=c0,yn=c0,
note_text(line_no=c0,text_line=c0)=c0%),
feos(label=c0,x=c0,y=c0,picture=c0)=c0%,
labels(data_id=c0,name=c0,x=c0,y=c0)=c0%,
)=c0%
) from nested_example.dat;
```

Erase Database

The following SQL/NF script eliminates all the data in the nested-relational implementation of the IDEF₀ database.

```
DELETE FROM project;
```

Show Database

The following SQL/NF script shows all the data in the nested-relational implementation of the IDEF₀ database.

```
SELECT ALL FROM project;
```

Extract Drawing Data

The following query extracts drawing data from the nested-relational implementation of the IDEF₀ database. The query is associated with the diagram shown in Figure 8.

```

SELECT (SELECT ALL BUT segments.data_id,labels.data_id FROM sheets WHERE node = "AO")
FROM project
WHERE project_name = "DM Example";

(sheets)
|c_number |node |name |author |version |date |
|-----|-----|-----|-----|-----|-----|
|2 |AO |manage database |Gerald R. Morris |1.0 |02/14/89 |
|-----|-----|-----|-----|-----|-----|
(boxes)
|node |name |x |y |visible_DRE |
|-----|-----|---|---|-----|
|A1 |manage numeric data |16 |16 | -1 |
|A2 |manage alpha data |17 |17 | -1 |
|-----|-----|---|---|-----|
(segments)
(location)
|xs |ys |xe |ye |
|---|---|---|---|
|18 |18 |19 |19 |
|---|---|---|---|
(symbols)
|x |y |type_symbol |symbol_type |
|-----|-----|-----|-----|
|18 |18 |boundary |I1 |
|-----|-----|-----|-----|
|19 |19 |21 |21 |
|---|---|---|---|
|21 |21 |arrow |right-arrow |
|-----|-----|-----|-----|
|19 |19 |23 |23 |
|-----|-----|-----|-----|
|19 |19 |turn |right-down |
|-----|-----|-----|-----|
|23 |23 |24 |24 |
|-----|-----|---|---|
|23 |23 |turn |down-right |
|24 |24 |arrow |right-arrow |
|-----|-----|-----|-----|
|26 |26 |27 |27 |
|-----|-----|---|---|
|26 |26 |boundary |C1 |
|27 |27 |turn |right-down |
|-----|-----|-----|-----|
|27 |27 |29 |29 |
|-----|-----|---|---|
|29 |29 |arrow |down-arrow |
|-----|-----|-----|-----|
|27 |27 |32 |32 |
|-----|-----|---|---|
|32 |32 |turn |right-down |
|-----|-----|-----|-----|

```

```

|---|---|---|---|
|32 |32 |33 |33 |
|---|---|---|---|
|---|---|-----|-----|
|33 |33 |arrow      |down-arrow   |
|---|---|-----|-----|
|---|---|---|---|
|35 |35 |36 |36 |
|---|---|---|---|
|---|---|-----|-----|
|35 |35 |boundary  |01
|35 |35 |arrow      |right-arrow  |
|---|---|-----|-----|
|---|---|---|---|
|38 |38 |39 |39 |
|---|---|---|---|
|---|---|-----|-----|
|39 |39 |turn      |right-down  |
|---|---|-----|-----|
|---|---|---|---|
|39 |39 |40 |40 |
|---|---|---|---|
|---|---|-----|-----|
|40 |40 |turn      |down-right  |
|---|---|-----|-----|
|---|---|---|---|
|40 |40 |36 |36 |
|---|---|---|---|
|---|---|-----|-----|
|---|---|-----|-----|
|---|---|---|---|
|42 |42 |43 |43 |
|---|---|---|---|
|---|---|-----|-----|
|43 |43 |turn      |right-up    |
|---|---|-----|-----|
|---|---|---|---|
|43 |43 |36 |36 |
|---|---|---|---|
|---|---|-----|-----|
|36 |36 |turn      |up-right    |
|---|---|-----|-----|
|---|---|---|---|
|40 |40 |41 |41 |
|---|---|---|---|
|---|---|-----|-----|
|40 |40 |arrow      |down-arrow  |
|---|---|-----|-----|
|---|---|---|---|
|75 |75 |76 |76 |
|---|---|---|---|
|---|---|-----|-----|
|75 |75 |tunnel     |hidden-source |
|76 |76 |to-from-all |A
|76 |76 |arrow      |right-arrow  |
|---|---|-----|-----|
|---|---|---|---|
|77 |77 |78 |78 |
|---|---|---|---|
|---|---|-----|-----|
|77 |77 |to-from-all |A
|78 |78 |arrow      |right-arrow  |
|---|---|-----|-----|

```

```

|---|---|---|---|
|79 |79 |80 |80 |
|---|---|---|---|
|---|---|-----|-----|
|79 |79 |to-from-all |A
|80 |80 |arrow      |right-arrow |
|---|---|-----|-----|
(squiggles)
|x1 |y1 |x2 |y2 |x3 |y3 |x4 |y4 |
|---|---|---|---|---|---|---|---|
|---|---|---|---|---|---|---|---|
(meta_notes)
|label |x |y |
|-----|---|---|
|-----|---|---|
(note_text)
|line_no |text_line      |
|-----|-----|
|-----|-----|
(foot_notes)
|label |xm |ym |xn |yn |
|-----|---|---|---|---|
|-----|---|---|---|---|
(note_text)
|line_no |text_line      |
|-----|-----|
|-----|-----|
(feos)
|label |x |y |picture      |
|-----|---|---|-----|
|-----|---|---|-----|
(labels)
|name      |x |y |
|-----|---|---|
|userdata |17 |17 |
|unumber  |20 |20 |
|ualpha   |22 |22 |
|rules    |25 |25 |
|numberrules |28 |28 |
|alpharules |30 |30 |
|feedback |34 |34 |
|numbermsgs |37 |37 |
|alphamsgs |41 |41 |
|fctrl/A13 |55 |55 |
|error code |85 |85 |
|-----|---|---|

```

Extract Essential Data

The following queries extract essential data from the nested-relational implementation of the IDEF₀ database. The first query extracts the data for a typical activity data dictionary. The second query extracts the data for a typical data element data dictionary. As before, these queries are associated with the diagram shown in Figure 8.

Activity Data Dictionary. This next SQL/NF query illustrates the extraction of the data dictionary data (essential data) associated with a typical activity (A1 in the example).

```

SELECT (SELECT ALL BUT node_id,c_number,hist_calls,children FROM activities WHERE node = "A1")
FROM project
WHERE project_name = "DM Example";

(activities)
|node |name          |author          |version |date      |changes      |parent      |
|----|-----|-----|-----|-----|-----|-----|
|A1  |manage numeric data |Gerald R. Morris |1.0    |12/14/89 |           |manage database |
|----|-----|-----|-----|-----|-----|-----|
(act_descr)
|line_no |descr_line          |
|-----|-----|
|1      |This activity will  |
|2      |handle numbers        |
|-----|-----|
(references)
|ref_type          |
|-----|
|contract          |
|-----|
(ref_lines)
|line_no |ref_line          |
|-----|-----|
|1      |KMR E00028-89-0123 3.3.2.1.2a |
|-----|-----|
(data_elems)
|data_name          |icom_type |
|-----|-----|
|unumber            |I          |
|errors             |I          |
|numberrules        |C          |
|numbermsgs         |O          |
|-----|-----|

```

Data Element Data Dictionary This next SQL/NF query illustrates the extraction of the data dictionary data (essential data) associated with a typical data element (unumber in the example).

```

SELECT (SELECT ALL BUT data_id FROM data_elements WHERE name = "unumber")
FROM project
WHERE project_name = "DM Example";

(data_elements)
|name      |author          |version |date      |changes |parent   |
|-----|-----|-----|-----|-----|-----|
|unumber   |Gerald R. Morris |1.0     |02/14/89 |        |userdata |
|-----|-----|-----|-----|-----|-----|
(data_descr)
|line_no  |descr_line      |
|-----|-----|
|1        |This is the user numeric data |
|-----|-----|
(references)
|ref_type |
|-----|
|AFM      |
|-----|
(ref_lines)
|line_no |ref_line      |
|-----|-----|
|1        |AFM 35-10 page 3 para. 2.3 |
|-----|-----|
(aliases)
|name      |where_used     |comment   |
|-----|-----|-----|
|-----|-----|-----|
(min_max)
|data_type |minimum       |maximum   |
|-----|-----|-----|
|-----|-----|-----|
(range)
|data_type |range         |
|-----|-----|
|integer   |integer'range |
|-----|-----|
(values)
|data_type |value         |
|-----|-----|
|-----|-----|
(activites)
|node_name      |icom_type |
|-----|-----|
|manage numeric data |I      |
|-----|-----|
(children)
|data_name      |
|-----|-----|
|-----|-----|

```

Appendix J. Ada Package for Drawing Data Structures

The following (incomplete) package specification illustrates the data structures that might be used to capture the drawing data in the IDEF₀ database. Obviously some type of embedded query language capability would be required.

```
with activity_data, data_element_data, analyst_data;
package drawing_data is

  -- This package defines the data structures that are used to capture the
  -- drawing data from the IDEF0 database via embedded query language calls
  --
  -- It is not known apriori how many tuples there are, so a linked list
  -- structure is used.

  -- The element names correspond identically to the attribute names used in
  -- the IDEF0 database. It is assumed the user of this package is familiar
  -- with the database schema...

  type box;
  type box_pointer is access box;
  type box is record
    node      : activity_data.node_type;
    name      : activity_data.name_type;
    x         : integer;
    y         : integer;
    visible_dre : integer;
    next_box  : box_pointer := null;           -- next box in list
  end record;

  type loc;
  type loc_pointer is access loc;
  type loc is record
    xs      : integer;
    ys      : integer;
    xe      : integer;
    ye      : integer;
    next_loc : loc_pointer := null;           -- next location
  end record;

  type symbol;
  type symbol_pointer is access symbol;
  type symbol is record
    x        : integer;
    y        : integer;
    type_symbol : string(1..12);
    symbol_type : string(1..12);
    next_symbol : symbol_pointer := null;      -- next symbol
  end record;
```

```

type seg;
type seg_pointer is access seg;
type seg is record
    location : loc_pointer := null;
    symbols : symbol_pointer := null;
    next_seg : seg_pointer := null;           -- next segment
end record;

type squig;
type squig_pointer is access squig;
type squig is record
    x1      : integer;
    y1      : integer;
    x2      : integer;
    y2      : integer;
    x3      : integer;
    y3      : integer;
    x4      : integer;
    y4      : integer;
    next_squig : squig_pointer := null;        -- next squiggle
end record;

type note_txt;
type note_txt_pointer is access note_txt;
type note_txt is record
    line_no      : integer;
    text_line    : string(1..60);
    next_note_txt : note_txt_pointer := null;    -- next line of text
end record;

type meta;
type meta_pointer is access meta;
type meta is record
    label      : string(1..1);
    x         : integer;
    y         : integer;
    note_text : note_txt_pointer := null;
    next_meta : meta_pointer := null;           -- next meta-note
end record;

type foot;
type foot_pointer is access foot;
type foot is record
    label      : string(1..1);
    xm        : integer;
    ym        : integer;
    xn        : integer;
    yn        : integer;
    note_text : note_txt_pointer := null;
    next_foot : foot_pointer := null;           -- next foot-note
end record;

```

```

type feo;
type feo_pointer is access feo;
type feo is record
    label      : string(1..1);
    x          : integer;
    y          : integer;
    picture    : string(1..60);
    next_feo   : feo_pointer := null;                      -- next FEO
end record;

type label;
type label_pointer is access label;
type label is record
    name       : string(1..10);
    x          : integer;
    y          : integer;
    next_label : label_pointer := null;                    -- next label
end record;

type sheet;
type sheet_pointer is access sheet;
type sheet is record
    c_number   : integer;
    node       : activity_data.node_type;
    name       : activity_data.name_type;
    author     : analyst_data.author_type;
    version    : activity_data.version_type;
    date       : activity_data.date_type;
    boxes      : box_pointer := null;
    segments   : seg_pointer := null;
    squiggles  : squig_pointer := null;
    meta_notes: meta_pointer := null;
    foot_notes: foot_pointer := null;
    feos       : feo_pointer := null;
    labels     : label_pointer := null;
end record;

procedure draw_a_0_sheet(the_sheet : in sheet_pointer);
    -- and some other stuff as well
end drawing_data;

```

Bibliography

1. Austin, Capt Kenneth A. *SAtool Interface to the SDI Architecture Dataflow Modeling Technique*. MS thesis, Air Force Institute of Technology, December 1989.
2. Bancilhon, Francois. "Object-Oriented Database Systems." *ACM SIGACT-SIGMOD SIGART PODS*, pages 152-162 (1988).
3. Carey, Michael, et al. "An Overview of the Exrel Relational DBMS." Computer Sciences Department, University of Wisconsin, 1989.
4. Carey, Michael, et al. "The EXODUS Extensible DBMS Project: An Overview." Computer Sciences Department, University of Wisconsin, 1989.
5. Carey, Michael, et al. "Using the EXODUS Storage Manager V1.2." Computer Sciences Department, University of Wisconsin, 1989.
6. Chen, P. Pin-Shan. "The Entity-Relationship Model-Toward a Unified View of Data." *ACM Transactions on Database Systems*, 1(1):9-36 (1976).
7. Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM*, 13(6):377-387 (1970).
8. Colby, Latha S. *A Recursive Algebra for Nested Relations*. Technical Report, Indiana University, January 1989.
9. Connally, Capt Ted D. *Common Database Interface for Heterogeneous Software Engineering Tools*. MS thesis, Air Force Institute of Technology, December 1987 (AD-A189628)
10. Date, C. J. *An Introduction to Database Systems*. Addison-Wesley Publishing Company, 1981.
11. DeMarco, Tom. *Structured Analysis and System Specification*. Prentice Hall, 1979.
12. Fairley, Richard E. *Software Engineering Concepts*. McGraw-Hill Book Company, 1985.
13. Forman, Betty Y. "SuperPDL Puts Software Systems Design On-Line." *Digital Review*, pages 53-55 (August 24 1987).
14. Goering, Richard. "Partnership Links CASE to Software Test." *Computer Design*, page 38 (May 1 1988).
15. Goering, Richard. "Standardization Effort Targets Data Management for CASE." *Computer Design*, page 28 (October 1 1988).
16. Hartrum, Thomas C. *System Development Documentation Guidelines and Standards* (Draft 4 Edition). Department of Electrical and Computer Engineering, Air Force Institute of Technology, January 2 1989.
17. Hawley, Sue Ann. "CASE For Sale." *DEC Professional*, pages 52-54 (December 1987).
18. Jensen, Randall, et al. "ESML: An Extended Systems Modeling Language Based on the Data Flow Diagram." Preliminary information distributed by Dr. Jensen in MPP RTA, Real Time Analysis. Hughes Aircraft Company, Ground Systems Group, Fullerton, California, November 2 1987.
19. Johnson, Capt Steven E. *A Graphics Editor for Structured Analysis with a Data Dictionary*. MS thesis, Air Force Institute of Technology, December 1987 (AD-A190618).
20. Korth, Henry F. and Abraham Silberschatz. *Database Systems Concepts*. New York, NY 10020 McGraw-Hill Book Company, 1986.
21. Lamont, Gary B. "An Introduction to Big-O and His Friends." Class handout for EENG 586, Advanced Information Structures, Fall Quarter 1988.

22. Makinouchi, A. "A Consideration of Normal Form of Not-Necessarily-Normalized Relations in the Relational Data Model," *Proc. 3rd VLDB*, pages 447-453 (1977).
23. Mankus, Capt Michael A. *Design and Implementation of the Nested Relational Data Model Under the Exodus Extensible Database System*. MS thesis, Air Force Institute of Technology, December 1989.
24. Marca, David A. and Clement L. McGowan. *SADT Structured Analysis and Design Technique* McGraw-Hill Book Company, 1988.
25. Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson AFB, OH 45433. *Integrated Computer-Aided Manufacturing (ICAM) Function Modeling Manual (IDEF₀)*, June 1981.
26. Ozsoyoglu, Meral Z. and Li-Yan Yuan. "A New Normal Form for Nested Relations," *ACM Transactions on Database Systems*, 12(4):111-136 (March 1987).
27. Pressman, Roger S. *Software Engineering: A Practitioner's Approach*. New York, NY 10020: McGraw-Hill Book Company, 1987.
28. Relational Technology (now Ingres Corporation), Inc., Alameda, California 94501. *INGRES/Embedded SQL User's Guide and Reference Manual*, 1986.
29. Relational Technology (now Ingres Corporation), Inc., Alameda, California 94501. *INGRES/SQL REFERENCE MANUAL*, 1986.
30. Ross, Douglas T. "Structured Analysis (SA): A language for Communicating Ideas," *IEEE Transactions on Software Engineering*, SE-3(1):16-34 (January 1976).
31. Roth, Capt Mark A. *Theory of Non-First Normal Form Relational Databases*. PhD dissertation, University of Texas at Austin, May 1986.
32. Roth, Mark, et al. "Extended Algebra and Calculus for Nested Relational Databases," *ACM Transactions on Database Systems*, 13(4):389-417 (December 1988).
33. Roth, Mark A., et al. "SQL/NF: A Query Language for \neg INF Relational Databases," *Information Systems*, 12(1):99-114 (1987).
34. Rubenstein, W. Bradley. "A Database Design for Musical Information," *ACM SIGMOD*, 16(3):479-490 (1987).
35. Smith, Capt Nealon F. *Implementation of SAtool II in Ada*. MS thesis, Air Force Institute of Technology, December 1989.
36. Stonebraker, Michael. *Readings in Database Systems*. San Mateo, CA: Morgan Kaufmann, 1988.
37. Technology, Index. "Index Technology Announces Excelerator CASE Link to Digital's New VAX CCD/Plus," *CASEnews*, page 4 (July/August 1988).
38. Thomas, S.J. and P.C. Fischer. "Nested Relational Structures." In by P. Kannellakis, Edited, editor, *Advances in Computing Research III, The Theory of Databases*, JAI Press, 1986.
39. Vizard, Michael. "Interface Brings CASE Tool Links Close to Reality," *Digital Review*, page 101 (March 21 1988).
40. Ward, Paul. "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing," *IEEE Transactions on Software Engineering*, SE-12(2):198-210 (February 1986).
41. Yourdon, Edward and Larry Constantine. *Structured Design*. New York, NY 10020: Yourdon Press, 1978.

Vita

Captain Gerald R. Morris [REDACTED] He graduated from high school in Norwalk, California, in 1972 and enlisted in the United States Air Force in May, 1973. He served 7 years as an electronic technician for a variety of communications equipment. He was then accepted under the Airman Education and Commissioning Program and attended The Ohio State University, from which he received the degree of Bachelor of Science in Electrical Engineering (summa cum laude) in December, 1982. Upon graduation he received a regular commission in the USAF through the USAF Officer Training School where he was a distinguished graduate. He then served as an electrical engineer at the Defense Contract Administration Services Plant Representative Office, Hughes Aircraft Company, Fullerton, California. In 1986 he received the National Contract Management Association's 1st Place Blanche Witte Award for specifying, designing, and building a database management system to track government contracts. He entered the School of Engineering, Air Force Institute of Technology, in May, 1988.

P [REDACTED]

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS					
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited					
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE							
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCE/ENG/90M-2		5. MONITORING ORGANIZATION REPORT NUMBER(S)					
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION					
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology (AU) Wright-Patterson AFB, OH 45433-6583		7b. ADDRESS (City, State, and ZIP Code)					
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Strategic Defense Initiative Organization	8b. OFFICE SYMBOL (If applicable) SDIO/S/PT	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER					
8c. ADDRESS (City, State, and ZIP Code) Room 1E149, The Pentagon Washington, D.C. 20301-7100		10. SOURCE OF FUNDING NUMBERS					
<table border="1"> <tr> <td>PROGRAM ELEMENT NO</td> <td>PROJECT NO.</td> <td>TASK NO</td> <td>WORK UNIT ACCESSION NO</td> </tr> </table>				PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO
PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO				
11. TITLE (Include Security Classification) A Comparison of a Relational and Nested-Relational IDEF0 Data Model							
12. PERSONAL AUTHOR(S) Gerald R. Morris, Captain, USAF							
13a. TYPE OF REPORT MS thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 1990 March	15. PAGE COUNT 186				
16. SUPPLEMENTARY NOTATION							
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) software engineering, database management systems databases, computer aided design, computer aided manufacturing					
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Thesis advisor: Mark A. Roth, Major, USAF Assistant Professor of Electrical and Computer Engineering							
20. DISTRIBUTION AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED					
22a. NAME OF RESPONSIBLE INDIVIDUAL Mark A. Roth, Major, USAF		22b. TELEPHONE (Include Area Code) (513)255-3576	22c. OFFICE SYMBOL AFIT/ENG				

(block 19 continued)

Abstract:

This thesis develops an abstract data model of a particular computer aided software engineering (CASE) methodology, and compares the query complexity, database size, and speed of query execution of a relational database management system (DBMS) implementation of the methodology with a nested-relational DBMS implementation of the same CASE methodology. In particular, the thesis considers the United States Air Force Integrated Computer Aided Manufacturing (ICAM) program's subset of Ross's Structured Analysis (SA) language called ICAM Definition Method Zero (IDEF₀). Ingres Corporation's relational DBMS, Ingres, is the implementation media for the relational version. The University of Wisconsin's extensible database, Exodus, is the implementation media for the nested-relational version.

The thesis provides background information on the development of CASE methodologies and the development of database management systems. Additionally, it provides an overview of the IDEF₀ analysis language, and the Exodus extensible DBMS.

Included in the thesis is an abstract data model of the IDEF₀ language. The model partitions IDEF₀ into an essential data model and a drawing data model. This partitioned representation facilitates ongoing and future research relative to syntax checking, generation of an executable software specification, and automatic layout of SA diagrams. Since IDEF₀ is the analysis methodology selected by the Strategic Defense Initiative Organization, the abstract data model alone is of importance.

The abstract data model is mapped into a relational representation and implemented within Ingres. The relational representation is mapped into a nested-relational representation and implemented within Exodus. The two implementations are compared to see if there are any advantages to be gained by using a nested-relational DBMS for this type of application (CASE tool data). The areas of comparison include query complexity, size . the database, and speed of query execution.